# Boid simulation

*Relazione progetto d'esame del corso Programmazione per la Fisica*

Caterina Pollini, Arianna Zironi, Francesca Zurli

3 Gennaio 2025

## Indice

# 1 Descrizione e dinamica del sistema

Il progetto ha come obiettivo la realizzazione di un programma che simuli il comportamento di uno stormo di uccelli in volo in uno spazio bidimensionale. Lo stormo è composto da due categorie di agenti: i boids, che rappresentano gli uccelli, e i predatori.
Ogni boid, caratterizzato da una velocità $\vec{v_{b_i}}$ e da una posizione $\vec{x_{b_i}}$, modifica la propria velocità seguendo tre regole fondamentali di volo:

- **Separazione**: evita le collisioni con i boids vicini. La velocità di separazione è calcolata come:

$$\vec{v_1} = -s \sum_{i \neq j} (\vec{x_{b_j}} - \vec{x_{b_i}}), \tag{1}$$

  dove $s$ è un opportuno fattore di separazione che determina l'intensità della repulsione. Questa regola si applica quando $|\vec{x_{b_i}} - \vec{x_{b_j}}| < d_s$, con $d_s$ distanza di separazione.

- **Allineamento**: orienta il boid nella stessa direzione dello stormo. La velocità di allineamento è definita da:

$$\vec{v_2} = a \left( \frac{1}{n-1} \sum_{i \neq j} \vec{v_{b_j}} - \vec{v_{b_i}} \right), \tag{2}$$

  dove $a$ è un fattore di allineamento che regola la rapidità con cui il boid si adatta alla direzione media dei vicini.

- **Coesione**: guida il boid verso il centro di massa dei boids vicini. La velocità di coesione è espressa come:

$$\vec{v_3} = c(\vec{x_c} - \vec{x_{b_i}}), \tag{3}$$

  dove $c$ è un fattore di coesione che determina la forza di attrazione verso il centro di massa $\vec{x_c}$ dei vicini.

Inoltre, ogni boid applica esclusivamente la regola di separazione nei confronti dei predatori, cercando di fuggire da questi ultimi. I predatori, invece, inseguono la preda a loro più vicina dello stormo.
Le equazioni sopra riportate governano l'evoluzione temporale dello stormo. Dalle interazioni tra gli agenti emerge un comportamento collettivo caratterizzato dall'ordine, una proprietà distintiva dei sistemi complessi adattivi (*complex adaptive systems*).

# 2 Scelte progettuali ed implementative

## 2.1 Struttura del codice

Il codice, scritto in linguaggio C++, è organizzato come segue:

- **Vector.cpp** e la relativa interfaccia **Vector.hpp**: definiscono una classe per la gestione di vettori bidimensionali, utilizzati per il calcolo vettoriale tra posizioni e velocità dei boids.

- **Boid.cpp** e la relativa interfaccia **Boid.hpp**: rappresentano il singolo boid. Contengono gli attributi di posizione, velocità e i metodi necessari a descrivere le tre regole di volo per ciascun boid.

- **Flock.cpp** e la relativa interfaccia **Flock.hpp**: gestiscono il comportamento dell'insieme dei boids come stormo, applicando nel tempo le regole di volo a ciascun componente.

- **Vector.test.cpp**: include i test per verificare il funzionamento della classe `Vector`.
- **Boid.test.cpp**: include i test per verificare il funzionamento della classe `Boid`.
- **Flock.test.cpp**: include i test per verificare il funzionamento della classe `Flock`.
- **main.cpp**: gestisce i messaggi di input e output, oltre all'implementazione della visualizzazione grafica del modello tramite la libreria SFML.

## 2.2 Dettagli di implementazione

### 2.2.1 Classi e struct

Per implementare il sistema sono state create tre classi e una struct:

- **Classe Vector**:
  - **Parte privata**: contiene una coppia di valori di tipo `float`, che rappresentano le coordinate di un punto nello spazio bidimensionale.
  - **Parte pubblica**: contiene i metodi necessari per effettuare operazioni vettoriali come somma, sottrazione, normalizzazione e distanza, fondamentali per le regole di volo dei boids.
- **Classe Boid**:
  - **Parte privata**: contiene due elementi di tipo `Vector`, che rappresentano la posizione e la velocità dei boids, un valore di tipo `float` che corrisponde all'angolo di visione entro cui vengono individuati i boids vicini e un oggetto di tipo `sf::CircleShape`, tratto dalla libreria grafica SFML, utilizzato per rappresentare graficamente i boids con forma triangolare.
  - **Parte pubblica**: contiene le definizioni delle tre regole di volo, che prendono come argomenti parametri forniti come input nel `main`, due metodi per garantire una velocità massima e una minima a ciascun boid, evitando così comportamenti indesiderati e metodi per ruotare e spostare i singoli oggetti di tipo `Boid` durante la rappresentazione grafica, utili per aggiornare le posizioni nel corso della simulazione.
- **Classe Flock**:
  - **Parte privata**: contiene i parametri che le tre regole di volo prendono come argomenti, oltre a due vettori di oggetti di tipo `Boid`, che rappresentano rispettivamente l'insieme dei boids simulati e dei predatori.
  - **Parte pubblica**: i metodi della classe prendono in input un intervallo di tempo generico e gestiscono la variazione di velocità e posizione di ciascun boid, applicando le tre regole e aggiornando l'evoluzione grafica del modello.
- **Struct Statistics**: contiene i dettagli relativi alla distanza media tra i boids e alle loro velocità medie, con le rispettive deviazioni standard, fornendo così una rappresentazione statistica dello stormo.

### 2.2.2 Main

Il file `main.cpp` ha il compito di eseguire la simulazione vera e propria, visualizzandola graficamente e gestendo i parametri in input e output. In particolare, si definisce una finestra grafica di tipo `sf::RenderWindow`, tramite l'utilizzo della libreria grafica SFML, che rappresenta il contesto visivo in cui si evolvono i boids. Si implementa la gestione degli eventi e del tempo per visualizzare dinamicamente l'evoluzione dello stormo. La gestione del tempo è affidata a un oggetto `sf::Clock`, che utilizza:

- il metodo `.restart()`, per azzerare il conteggio del tempo a ogni frame;
- il metodo `.asSeconds()`, per restituire il tempo trascorso tra due frame consecutivi in secondi.

### 2.2.3 Comportamento ai bordi

Un aspetto cruciale della simulazione è la gestione del comportamento dei boids ai bordi dell'area di simulazione.
Inizialmente, l'implementazione prevedeva che i boids rimanessero all'interno dei bordi: quando uno di essi raggiungeva un bordo, la sua velocità veniva invertita di segno, simulando una sorta di "rimbalzo".
Tuttavia, con l'introduzione dei predatori, questa regola ha mostrato alcune criticità: nei pressi dei limiti della finestra grafica, la tendenza dei boids a fuggire dai predatori si scontrava con la regola dell'inversione di velocità, generando comportamenti incoerenti nei casi in cui un boid raggiungesse il bordo e fosse contemporaneamente seguito da un predatore.
Per risolvere questo problema, è stata adottata una soluzione alternativa: nella nuova implementazione, i boids e i predatori che oltrepassano l'area di simulazione vengono traslati al lato opposto della finestra grafica, seguendo una logica di spazio bidimensionale toroidale. Questa modifica ha migliorato significativamente la fluidità e la coerenza della simulazione.

## 2.3 Repository su GitHub

Il progetto è disponibile su GitHub al seguente indirizzo `https://github.com/catepollini/Progetto-2024`. Poiché il progetto è stato sviluppato in gruppo, l'utilizzo di GitHub si è rivelato fondamentale per favorire la collaborazione tra i membri e gestire in modo efficace gli sviluppi e le modifiche al codice.

# 3 Istruzioni su come compilare, testare ed eseguire

Inizialmente, è necessario scaricare i file inerenti alla libreria grafica utilizzata per rappresentare la simulazione, SFML, attraverso il seguente comando: `sudo apt install libsfml-dev`. Successivamente si può proseguire con l'esecuzione del programma e dei test. Il programma contiene un file di configurazione `CMakeLists.txt`. Una volta aperto il terminale si digitano i seguenti comandi per compilare il codice sorgente e generare l'eseguibile:

- `cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug`;
- `cmake --build build`.

Successivamente è possibile eseguire il programma principale attraverso il comando `./build/boids` e i vari test unitari attraverso `./build/vector.t`, `./build/boid.t`, `./build/flock.t` .

# 4 Input e Output

- **Input**

  All'avvio della simulazione, il programma richiede l'inserimento di 5 parametri tramite standard input. Questi parametri influenzano il comportamento dello stormo e dei predatori. Per ciascun parametro sono specificati i valori minimi e massimi accettati, in modo tale che i valori inseriti siano ottimali e garantiscano una simulazione più realistica.

  - `d`: rappresenta la distanza entro cui due boids sono considerati interagenti, applicando le tre regole di volo. Il valore deve essere scelto in modo da consentire ai boids di rilevare i vicini più prossimi: valori troppo alti o troppo bassi potrebbero compromettere la formazione dello stormo. Il range accettabile è [60; 200].
  - `ds`: rappresenta la distanza di separazione, entro cui si applica la regola di separazione. Questo valore deve essere inferiore a `d` per garantire che i boids rimangano vicini al proprio stormo senza sovrapporsi. Il range accettabile è [30; 50].
  - `s`: rappresenta il fattore di separazione, utilizzato nella legge di separazione. Il valore deve essere scelto nel range [0.3; 0.5].
  - `a`: rappresenta il fattore di allineamento, utilizzato nella legge di allineamento. Il range accettabile è [0.4; 0.8].
  - `c`: rappresenta il fattore di coesione, utilizzato nella legge di coesione. Il valore può variare nel range [0.0001; 0.0004].

  Dopo l'avvio della finestra grafica, l'utente può interagire con la simulazione tramite mouse o touchpad:

  - Premendo il tasto sinistro del mouse, viene aggiunto un boid alla simulazione.
  - Premendo il tasto destro del mouse, viene aggiunto un predatore.

  Entrambi gli agenti vengono generati con una posizione determinata dal punto di clic del mouse o touchpad e con componenti della velocità iniziale casuali estratte nel range [-100; 100] da una distribuzione uniforme. Inoltre, viene generato l'angolo di visione estratto nel range [120; 180] da una distribuzione uniforme.
  Per distinguere i boids dai predatori, viene utilizzato il metodo `.setFillColor()` della libreria grafica SFML: i boids sono rappresentati con il colore verde, mentre i predatori con il colore rosso.
  Sebbene non vi siano limiti al numero massimo di boids e predatori da aggiungere, si consiglia di non eccedere per evitare rallentamenti nella simulazione e una rappresentazione troppo caotica.

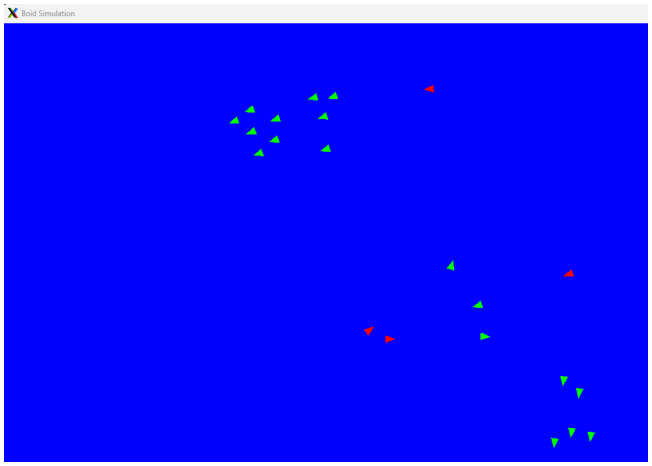


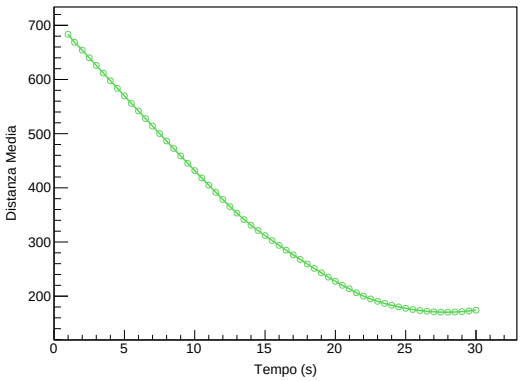

Figura 1: *Esempio di come appare la finestra grafica durante la simulazione.*

- **Output**

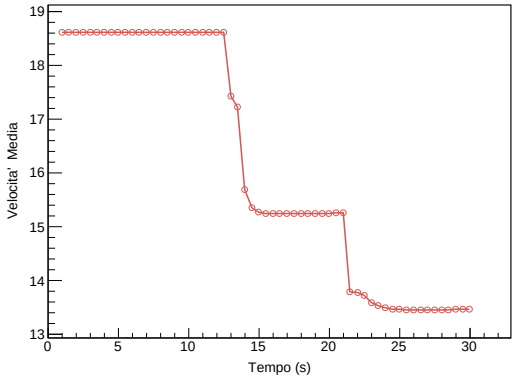
  Tramite il metodo `state` della classe `Flock`, il programma restituisce in output parametri descrittivi del comportamento collettivo dello stormo, quali la distanza media tra i boids nel tempo e la velocità media (in modulo) dei boids nel tempo. Si restituiscono i valori con le rispettive deviazioni standard. I risultati sono stampati sul terminale ogni due secondi, con la condizione che siano presenti almeno 2 boids per garantire una statistica significativa.

## 5 Interpretazione dei risultati ottenuti

Per verificare che la simulazione riproducesse il comportamento atteso, è stato realizzato un programma di simulazione, `simulation.cpp`, inizializzato con 5 boids aventi velocità e posizioni arbitrarie. Queste posizioni sono state scelte in modo da favorire la convergenza verso il centro della finestra, evitando comportamenti indefiniti ai bordi. Il programma ha generato in output un file di dati, `simulation_data.cpp`, contenente informazioni sul tempo trascorso, oltre alle distanze e velocità medie, registrate ogni due secondi.
I dati raccolti sono stati analizzati utilizzando il framework ROOT. In particolare, è stata sviluppata la macro `analyze.C` per visualizzare i risultati tramite i grafici riportati di seguito.




(a) Andamento della distanza media.




(b) Andamento della velocità media.

Figura 2: *Grafici realizzati con il framework ROOT: sulle ascisse è riportato il tempo, misurato in secondi; sulle ordinate sono rappresentati rispettivamente l'andamento della distanza media (a) e della velocità media (b).*

Il primo grafico mostra come la distanza media tra i boids diminuisca progressivamente fino a stabilizzarsi, raggiungendo il limite minimo imposto dalla regola di separazione.
Il secondo grafico evidenzia che la velocità media tende a stabilizzarsi verso un valore costante. Questo comportamento emerge con il progressivo avvicinamento reciproco dei boids e la conseguente formazione dello stormo.
I risultati ottenuti sono coerenti con le previsioni teoriche, indicando che il sistema simulato è coeso, dinamico e sensibile ai parametri impostati. Essi confermano l'efficacia delle regole implementate e la capacità del modello di replicare comportamenti realistici degli stormi.

# 6 Strategia di test

Per verificare il corretto funzionamento dei metodi definiti all'interno del programma, è stata utilizzata la libreria DOCTEST, inclusa tramite il file `doctest.h`. In questo modo è stato possibile creare situazioni specifiche per controllare che le diverse componenti del programma restituissero i risultati attesi, garantendo così l'affidabilità del codice. In alcuni casi è stata impiegata la funzionalità `doctest::Approx`, particolarmente utile per gestire valori restituiti di tipo `float`. Questa opzione consente di specificare un'approssimazione accettabile per i risultati.
I dettagli relativi ai test sono documentati nei file il cui nome termina con `test.cpp`.
Alcuni metodi non erano adatti all'utilizzo di DOCTEST, in particolare quelli relativi alla grafica, come i metodi che definiscono le forme e i movimenti dei boids. La correttezza di queste funzioni è stata verificata durante l'esecuzione del programma, controllando che i boids rispettassero i requisiti dell'implementazione. Si è verificato che i boids si muovessero correttamente nello spazio toroidale. Dopo numerosi test empirici, la rappresentazione grafica della simulazione è stata ritenuta soddisfacente, confermando la validità delle funzioni implementate.
Infine, per garantire la formattazione del codice, è stato utilizzato Clang-Format, configurato tramite un file `.clang-format`.