Simulazione del comportamento di uno stormo

Cecilia Pacetti, Sara Montelli

6 Gennaio 2026

Indice

1 Descrizione del sistema

Il progetto ha come obiettivo la realizzazione di un programma in C++ che simuli il comportamento collettivo di uno stormo di uccelli all'interno di uno spazio bidimensionale. La simulazione si basa sulla dinamica interattiva tra due diverse popolazioni: i boids (prede) e i predatori. Ogni boid è definito da un vettore posizione e da un vettore velocità unito a un attributo booleano che ne definisce il ruolo (preda o predatore). L'evoluzione dinamica del sistema emerge spontaneamente dalle interazioni locali tra i singoli elementi, come avviene nei sistemi complessi adattivi. Nello specifico, ogni boid aggiorna il proprio stato seguendo tre regole fondamentali:

- **Separazione:** permette di mantenere una distanza di sicurezza minima dagli altri membri dello stormo.
- **Allineamento:** spinge il boid ad adeguare la propria direzione a quella media dei compagni nel suo raggio di interazione.
- **Coesione:** agisce come una forza attrattiva verso il centro di massa locale dei boids vicini. Questa componente garantisce che lo stormo rimanga compatto, contrastando la tendenza dei singoli individui a disperdersi nello spazio.

Alle classiche regole, è stata aggiunta una dinamica predatore-preda asimmetrica. I boids sono dotati di una componente aggiuntiva della velocità, che riproduce la fuga: quando rilevano un predatore entro il raggio d'azione, applicano una forza di separazione per allontanarsi rapidamente dal predatore. Contemporaneamente, i predatori individuano il boid più vicino e si dirigono verso di esso per intercettarlo. L'ambiente è implementato in uno spazio toroidale che permette una simulazione continua senza confini fisici.

2 Scelte progettuali e implementative

Il progetto è disponibile su GitHub all'indirizzo: `https://github.com/saramontelli/progetto`.

2.1 Struttura del codice

Il progetto è sviluppato in C++ seguendo un approccio orientato agli oggetti, suddividendo le responsabilità del simulatore in componenti modulari:

- `vector.cpp` e il relativo header file `vector.hpp`: Questi file implementano la classe `math::Vector` che è il nucleo matematico del sistema. Si occupano della gestione delle coordinate 2D e di tutte le operazioni algebriche necessarie (somma, sottrazione, prodotto scalare, norma...). Una caratteristica peculiare è la gestione delle distanze in uno spazio toroidale tramite il metodo `shortest_delta`, fondamentale per permettere ai boids di interagire correttamente attraverso i bordi dello schermo.
- `boid.cpp` e il relativo header `boid.hpp`: Definiscono l'entità singola della simulazione. La classe `Boid` racchiude lo stato cinematico (posizione e velocità) e la logica individuale dell'agente. Qui sono implementate le leggi che governano il comportamento collettivo: separazione, allineamento e coesione, oltre ai meccanismi di reazione alla presenza di predatori.
- `flock.cpp` e il relativo header `flock.hpp`: Questo modulo fa da supervisore della simulazione. La classe `Flock` aggrega le istanze di boids e predatori, gestendo l'evoluzione temporale dell'intero sistema. Si occupa inoltre di aggiornare le velocità collettive, gestire le interazioni preda-predatore e calcolare i dati statistici tramite `FlockStats`.

- `main.cpp`: Si occupa dell'interazione con l'utente per la configurazione dei parametri fisici, inizializza l'ambiente di simulazione e gestisce il ciclo di rendering grafico utilizzando la libreria SFML.
- `vector.test.cpp`, `boid.test.cpp`, `flock.test.cpp`: Questi file implementano test unitari per verificare la correttezza delle rispettive classi.

2.2 Dettagli di implementazione

2.2.1 Classi e Struct

La simulazione si basa sull'interazione tra tre classi principali e una struct:

- **Classe `Vector`:**
  - **Parte privata:** incapsula due variabili di tipo `float` (x e y) che definiscono le componenti del vettore nello spazio bidimensionale.
  - **Parte pubblica:** espone l'interfaccia per il calcolo vettoriale. Oltre agli operatori aritmetici fondamentali, include metodi come `norm()` per il calcolo del modulo, `distance()` per la separazione spaziale e `shortest_delta()` per gestire correttamente le distanze attraverso i bordi dello spazio toroidale.
- **Classe `Boid`:**
  - **Parte privata:** contiene lo stato fisico dell'agente attraverso due oggetti `Vector` (posizione e velocità) e un flag booleano che ne identifica la natura (preda o predatore);
  - **Parte pubblica:** implementa la logica comportamentale del singolo boid. Include le funzioni per calcolare i vettori di separazione, allineamento e coesione. Fornisce inoltre i metodi per il `wrap_position` (rientro dai bordi) e lo `speed_limit`, garantendo che la velocità rimanga entro un intervallo cinematico ragionevole.
- **Classe `Flock`:**
  - **Parte privata:** funge da contenitore globale memorizzando i coefficienti per le regole di volo e i limiti di velocità. Organizza la popolazione della simulazione in due contenitori `std::vector` distinti per boids e predatori.
  - **Parte pubblica:** espone i metodi per l'evoluzione dinamica del sistema. La funzione `flock_update` coordina il movimento collettivo applicando le leggi fisiche a ogni componente, mentre `state` analizza l'intero stormo per generare dati statistici.
- **Struct `FlockStats`:** memorizza i valori medi e le deviazioni standard della velocità e della distanza tra i boids, permettendo di monitorare in tempo reale il grado di ordine e coesione dello stormo durante la simulazione.

2.2.2 Main

Costituisce il punto di ingresso del programma. Si occupa di acquisire i parametri dall'utente e inizializzare la finestra grafica tramite SFML. La gestione temporale è affidata a `sf::Clock` che calcola `delta_t` tra i frame, permettendo un movimento più fluido e indipendente dal frame rate. I boids vengono disegnati come triangoli ruotati in base alla direzione delle velocità. Gestisce inoltre gli input da tastiera per l'inserimento dinamico di nuovi boids (tasto B) o predatori (tasto P) e stampa periodicamente le statistiche della simulazione sul terminale.

2.2.3 Comportamento ai bordi

Per evitare che la simulazione sia confinata in un'area limitata, il progetto implementa uno spazio periodico toroidale. La funzione `wrap_position` garantisce che, quando un boid attraversa un bordo, venga riposizionato sul lato opposto. Grazie a `shortest_delta`, la distanza minima tra due boids viene calcolata considerando il percorso più breve attraverso i bordi, permettendo agli agenti di reagire correttamente ai compagni vicini anche dal margine opposto. In questo modo la simulazione risulta più fluida e coerente.

3 Istruzioni su compilazione, testing ed esecuzione

Il progetto si appoggia alla libreria grafica SFML per la visualizzazione della simulazione. Per installare la libreria, su sistemi basati su Ubuntu, si può usare il comando: `sudo apt install libsfml-dev`.

Il progetto contiene un file `CMakeLists.txt` per la configurazione e la compilazione. Per creare l'eseguibile e i test seguire questi passaggi da terminale:

- `cmake -S . -B build -G"Ninja Multi-Config"`
- `cmake --build build --config Debug`
- `cmake --build build --config Release`

Dopo la compilazione è possibile avviare il programma con:`./build/Debug/progetto` oppure `./build/Release/progetto`. Il progetto include tre test unitari:

- `vector_test`
- `boid_test`
- `flock_test`

I test possono essere eseguiti singolarmente con: `./build/Debug/boid_test`, `./build/Debug/vector_test`, `./build/Debug/flock_test` oppure nella versione Release con: `./build/Release/boid_test`, `./build/Release/vector_test`, `./build/Release/flock_test`.

4 Input e Output

4.1 Input e interazione

L'avvio della simulazione prevede una fase iniziale di configurazione tramite terminale, seguita da un'interazione dinamica una volta aperta la finestra grafica.

Configurazione da terminale: all'apertura del programma, l'utente deve definire la popolazione iniziale e cinque parametri fondamentali che regolano la fisica dello stormo. Per garantire una simulazione fluida e visivamente coerente il programma limita i parametri a range ottimali per ciascun valore:

- **Popolazione:** sono richiesti due numeri interi che rappresentano il numero iniziale di boids e predatori, i primi sono limitati a [1,60] mentre i secondi a [0,5]. Per distinguere gli agenti si utilizza il metodo `setFillColor()` di SFML in modo tale che i boids siano rappresentati con triangoli di colore azzurro, mentre i predatori con triangoli di colore rosso.
- **Closeness parameter (d):** definisce il raggio di interazione entro il quale un boid percepisce i propri vicini. Limitato al range [200,300];

- **Distance of separation (d_s):** distanza minima di sicurezza per evitare collisioni tra boids. Limitato al range [30,45];
- **Separation parameter (s):** influenza l'intensità della forza di repulsione quando i boids si avvicinano. Limitato al range [0.6,1.2];
- **Alignment parameter (a):** fa in modo che i boids procedano nella stessa direzione dello stormo. Limitato al range [0.04,0.09];
- **Cohesion parameter (c):** impone ai boids di avvicinarsi al centro di massa dei vicini. Limitato al range [0.005,0.01].

Interazione su Interfaccia Grafica: Una volta lanciata la simulazione, l'utente può modificare la popolazione in tempo reale tramite tastiera. Premendo il tasto B oppure il tasto P si aggiungono rispettivamente un nuovo Boid o un nuovo Predatore. Il programma include un controllo che limita il numero massimo di ciascuna specie secondo i range indicati, garantendo la stabilità del sistema.

La simulazione inizializza ogni agente con posizioni e velocità generate casualmente tramite distribuzioni uniformi. La coordinata x della posizione è compresa tra 0 e 1200 mentre la componente y può variare tra 0 e 800. Le velocità sono invece limitate al range [-110,110].

Esempio di configurazione: Al fine di agevolare la simulazione si riporta una configurazione esemplificativa che genera uno stormo stabile:

- **Popolazione:** 40 boids e 2 predatori
- **Parametri fisici:** $d = 250, \quad d\_s = 35, \quad s = 1.0, \quad a = 0.08, \quad c = 0.008.$

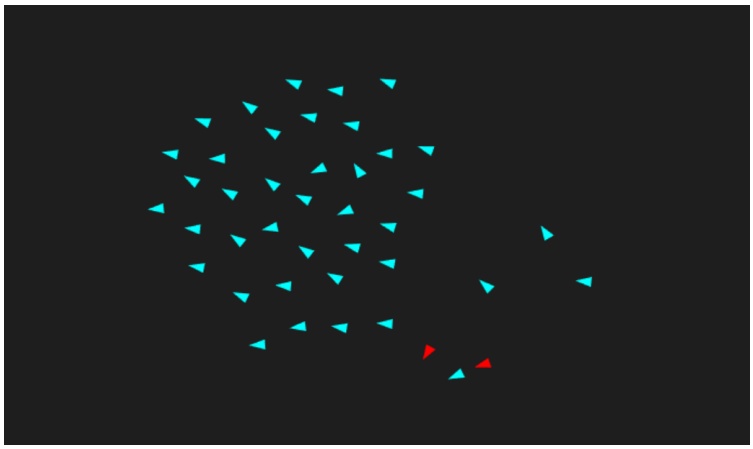


Figura 1: *Esempio di come appare la finestra grafica durante la simulazione.*

4.2 Output

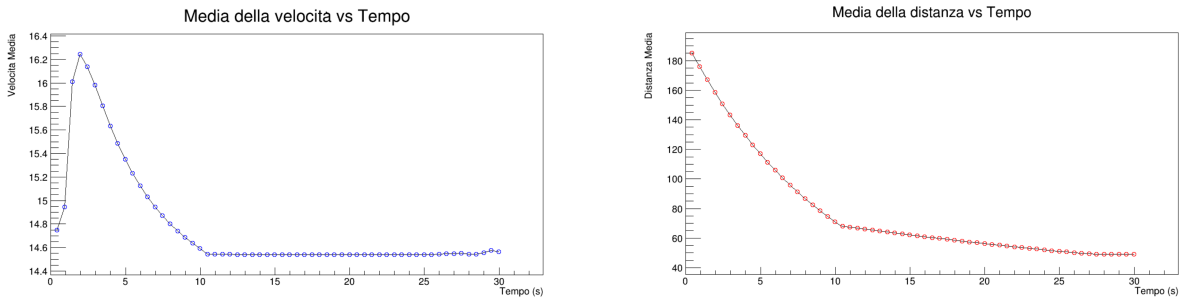
Il programma, tramite il metodo `state` della classe `Flock`, fornisce in output alcuni parametri descrittivi del comportamento collettivo dello stormo. In particolare viene calcolata la distanza media tra i boids accompagnata dalla relativa deviazione standard, lo stesso vale per il modulo della velocità media nel tempo. I risultati vengono stampati sul terminale ogni 100 step della simulazione.

5 Interpretazione dei risultati

Per verificare che il modello riproduca il comportamento collettivo atteso, è stato sviluppato un programma di simulazione dedicato (`flock_simulation.cpp`). La simulazione è inizializzata con cinque boids, le cui posizioni e velocità sono assegnate manualmente in modo da favorire una dinamica di avvicinamento verso una configurazione centrale ed evitare effetti indesiderati legati ai bordi del dominio. L'evoluzione temporale del sistema viene calcolata integrando le equazioni del moto con un passo temporale costante. A intervalli regolari di 0.5 secondi vengono calcolate, tramite il metodo `state` della classe `Flock`, le grandezze statistiche globali, in particolare la distanza media tra i boids e il modulo della velocità media. I valori ottenuti vengono salvati su file `flock_simulation.txt` e successivamente utilizzati per l'analisi grafica dei risultati grazie al framework ROOT.

Il primo grafico (Figura 2a) mostra l'andamento della velocità nel tempo, si osserva una fase transitoria iniziale in cui i boids cominciano a interagire. Grazie alle regole implementate, si osserva una rapida convergenza verso un valore di velocità uniforme. Il secondo grafico (Figura 2b) mostra l'andamento delle posizioni dei boids nel tempo. Dopo una fase di assestamento, la distanza media si stabilizza attorno a un valore critico, indicando la formazione di uno stormo coeso.

In sintesi, la convergenza di entrambi i grafici verso valori costanti rispecchia il comportamento teorico atteso e dimostra la capacità del modello di riprodurre dinamiche realistiche di comportamento collettivo.



(a) *Andamento della velocità media dei boids nel tempo* (b) *Andamento della distanza media dei boids nel tempo*

Figura 2: *Andamento delle grandezze statistiche dello stormo.*

6 Strategia di test

Per verificare il corretto funzionamento dei metodi implementati nel programma, è stata utilizzata la libreria di testing `DOCTEST`, inclusa tramite il file `doctest.h`. Quest'ultima ha permesso di costruire casi di test mirati, utili a controllare che le diverse componenti del codice restituissero i valori attesi. In particolare, per il confronto di grandezze di tipo `float`, è stata impiegata la funzionalità `doctest::Apprx` che consente di tollerare piccole differenze numeriche dovute all'aritmetica in virgola mobile. I dettagli relativi ai test unitari sono riportati nei file il cui nome termina con `test.cpp`. Alcuni metodi del programma, in particolare quelli legati alla gestione della grafica e alla visualizzazione della simulazione, non risultano adatti a essere testati tramite `DOCTEST`. La correttezza di tali funzionalità è stata quindi verificata durante l'esecuzione del programma, osservando il comportamento dei boids e dei predatori e controllando che il loro movimento e la loro interazione rispettassero i requisiti previsti, inclusa la corretta gestione dello spazio toroidale. A seguito di numerose verifiche empiriche, la rappresentazione grafica della simulazione è stata ritenuta soddisfacente.

Infine, per garantire una formattazione uniforme e leggibile del codice sorgente, è stato utilizzato Clang-Format, configurato tramite l'apposito file `.clang-format`.