

# Boids - Simulazione del volo di uno stormo di uccelli

DIFA UniBo - Programmazione per la Fisica

Michele Busti, Virginio Miroglio, Simone Pasquini, Giacomo Pivetti

Anno Accademico 2021/2022

## Indice

<b>1</b>	<b>Scelte progettuali e implementative</b>	<b>3</b>
<b>2</b>	<b>Istruzioni di compilazione, testing ed esecuzione</b>	<b>4</b>
2.1	Note aggiuntive . . . . .	5
<b>3</b>	<b>Formato di input e output</b>	<b>6</b>
<b>4</b>	<b>Interpretazione dei risultati ottenuti</b>	<b>6</b>
<b>5</b>	<b>Strategie di test</b>	<b>7</b>
<b>6</b>	<b>Conclusioni</b>	<b>8</b>
<b>7</b>	<b>Modifiche intercorse dalla consegna del progetto precedente</b>	<b>8</b>

# 1 Scelte progettuali e implementative

Il progetto è suddiviso in vari header file e file di implementazione in modo da gestire e suddividere al meglio i blocchi di base necessari per il corretto funzionamento della simulazione. In particolare, il codice si divide in:

- **Objects:** definisce classi e struct utili all'implementazione, come ad esempio i vettori 2-dimensionali per rappresentare posizioni e velocità dei boids, e con loro i vari metodi e le funzioni libere utili a gestirne alcuni calcoli matematici.
- **Functions:** definisce tutte le funzioni inerenti l'evoluzione dello stormo (come le regole di volo), quelle necessarie a una corretta evoluzione legate all'implementazione dei file "evolve" (come la funzione `influence`) e quelle utili al calcolo degli output richiesti (distanza e modulo della velocità media, con relative deviazioni standard).
- **Evolve:** definisce le funzioni che rendono possibile la simulazione dell'evoluzione dello stormo nel tempo. In particolare, si è scelto di definire due funzioni, una per l'evoluzione del singolo boid, l'altra per l'evoluzione dell'intero stormo (rispettivamente `evolve_boid` ed `evolve_flock`). In questo modo, oltre a una maggiore pulizia e linearità dal punto di vista logico, è stato estremamente più semplice gestire l'evoluzione dei singoli boids nel caso in cui questi ultimi si trovassero in situazioni particolari (come avere velocità nulla o essere in una posizione troppo vicina al predatore). Riguardo ai bordi della simulazione, è stato scelto di riposizionare ogni boid che superasse tale limite riposizionandolo specularmente su quello opposto (lo spazio è *bidimensionale toroidale*).
- **Sfml objects:** definisce due oggetti estremamente utili per l'implementazione grafica della simulazione, avvenuta attraverso la libreria multimediale SFML. In particolare, sono state importanti le creazioni di una classe `button`, che rende significativamente più agevole creare dei veri e propri pulsanti in SFML, e di una classe `data`, che consente la visualizzazione dei dati in output in maniera costantemente aggiornata.
- **Sfml:** descrive un'unica funzione (uno dei motivi per cui si è scelto di non dividere questa parte in header file e file di implementazione): `run_simulation`. Tale funzione accetta in input i valori necessari a creare la simulazione di uno stormo di boids, crea tutti gli oggetti di SFML necessari a una corretta visualizzazione grafica dei pulsanti,

dati di output, rappresentazione grafica di boid e predatore; infine, renderizza la finestra grafica con tutti gli oggetti appena citati.

- **Main:** illustra il codice necessario ad accettare e controllare i dati inseriti in input, quali ad esempio il numero di boids iniziali e i parametri per le regole di volo; nel codice del main è inclusa la funzione `run_simulation`, che consente di visualizzare la simulazione. I boids vengono generati secondo una distribuzione gaussiana che restituisce velocità e posizioni iniziali appropriate, seguendo le regole imposte ai bordi della finestra grafica e altri criteri opportunamente definiti, come la velocità massima e minima degli stessi.

## 2 Istruzioni di compilazione, testing ed esecuzione

La compilazione dei file sorgente è avvenuta tramite l'utilizzo del "meta" build system CMake.

Dalla directory `progetto`, estratta dal file `.../progetto.zip`, è possibile inserire il comando:

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

con cui vengono configurate le istruzioni di compilazione (la modalità `DEBUG` aggiunge l'opzione `-fsanitize=address` fra le direttive di compilazione, oltre a `-Wall -Wextra`).


Quindi, viene creata la directory `progetto/build` contenente le istruzioni necessarie a CMake per compilare correttamente i file che rappresentano il codice sorgente. In `progetto`, tramite il comando:



```
$ cmake --build build
```

vengono creati l'eseguibile `progetto/simulation` e i relativi test in `progetto/simulation.t`.

Eseguendo il file `progetto/simulation` viene richiesto l'inserimento del numero iniziale di boids e dei parametri  $s$ ,  $a$ ,  $c$  (separazione, allineamento e coesione) relativi alle tre regole di volo. In seguito, si visualizza la finestra grafica contenente i boids (esagoni arancioni) e un predatore fermo (quadrato nero). In alto a destra sono mostrati gli output richiesti (andamento nel tempo della distanza media e velocità media tra i boids con le rispettive deviazioni standard) costantemente aggiornati, mentre in alto a sinistra sono presenti tre pulsanti attraverso i quali è possibile aggiungere boids (Add boid) o rimuoverli (Remove boid) dalla simulazione (il numero minimo di






boids impostabili è due, infatti se tale condizione non fosse rispettata, non avrebbe più senso la rilevazione degli output desiderati) e un pulsante che permette di bloccare momentaneamente l'evoluzione della simulazione al fine di visualizzare i dati di output in un determinato istante (*Pause evolution*).


Premendo il tasto  è possibile mettere in movimento / fermare il predatore e, attraverso i tasti freccia della tastiera, se ne cambia la direzione e il verso del moto lungo gli assi del piano cartesiano.

Il tasto  aumenta la velocità del predatore,  la diminuisce. La capacità di incrementare la velocità del predatore è tale da consentire il raggiungimento dei boids stessi (in quanto è previsto l'effetto di separazione non solo tra i boids, ma anche tra boids e predatore) e la loro eliminazione nel momento in cui vengono intercettati (ogni variazione del numero di boids è registrata dal contatore presente in alto a sinistra).

Una volta arrivati a uno stormo di due soli boids, il predatore non sarà più in grado di eliminarne altri, per le stesse motivazioni fornite per il pulsante *Remove boid*.

Il predatore è in grado di eliminare i boids dallo stormo solamente dopo due secondi dall'inizio della simulazione (le motivazioni sono spiegate nelle *Note aggiuntive*). Quando la funzionalità di eliminazione sarà attivata, il predatore si colorerà di blu.

Inoltre, è possibile incrementare / diminuire a step di 0.005 i parametri  $s$ ,  $a$ ,  $c$ . Premendo i relativi pulsanti della tastiera (, , ) viene selezionato uno dei 3 parametri (evidenziato tramite una freccetta che indica il parametro attualmente selezionato); successivamente, si possono usare i tasti  per aumentare il parametro precedentemente selezionato,  per diminuirlo.

Durante la simulazione, premendo il tasto , vengono visualizzate tutte le istruzioni qui descritte.

## 2.1 Note aggiuntive

1. È possibile che il numero iniziale dei boids differisca di uno o due da quello inserito; ciò è dovuto al fatto che, essendo generati casualmente, alcuni boids potrebbero avere una velocità troppo elevata in direzione del predatore (inizialmente fermo). Di conseguenza, la velocità di separazione del boid che sta procedendo verso il predatore non è in grado di contrastare l'avanzamento dello stesso, che intercettando il predatore

viene da questi eliminato. Quindi, per ovviare a questo problema, si è definito che il predatore può eliminare i boids solo dopo un certo lasso di tempo dall'inizio della simulazione, sufficiente a farne stabilizzare le velocità.

2. Compilando con l'opzione `-fsanitize=address`, una volta chiusa la finestra grafica (e terminato quindi il programma) vengono restituiti dal terminale dei messaggi di errore riguardanti dei memory leak. A seguito di una ricerca approfondita, si è giunti alla conclusione che gli errori restituiti sono in realtà molto comuni e non dovuti a problematiche del codice sorgente, ma molto probabilmente a incompatibilità fra driver grafici e SFML. Difatti, nella fase di esecuzione che precede l'avvio della simulazione, non vengono comunque restituiti messaggi di errore riguardanti memory leak.

### 3 Formato di input e output

Al fine di ottenere una corretta compilazione, è necessario fornire come dati in input il numero iniziale di boids, di tipo `int`, e i tre parametri  $s$ ,  $a$ ,  $c$ , di tipo `double`. Per ottimizzare la visualizzazione grafica, è preferibile inserire un numero iniziale di boids minore di 150 e i seguenti valori per i tre parametri:  $s = 0.07$ ,  $a = 0.06$ ,  $c = 0.015$ .

La visualizzazione dell'output della simulazione, che come detto corrisponde ai dati inerenti distanze e velocità, avviene attraverso la rappresentazione di un oggetto testo di SFML nella finestra grafica che viene aggiornato automaticamente ogni due secondi con nuovi dati. Inoltre, come già descritto, è con il pulsante `Pause evolution` che si può fermare temporaneamente l'evoluzione dello stormo, al fine di visualizzare i dati della simulazione relativi a un certo istante di tempo.

### 4 Interpretazione dei risultati ottenuti

Per un'interpretazione ottimale dei risultati si può analizzare il comportamento che i boids hanno nella simulazione visualizzata grazie all'uso di SFML. Quello che inizialmente è un'insieme disordinato, in un breve numero di evoluzioni si trasforma in uno o più stormi che, incontrandosi a loro volta, si uniformano andando a creare un unico grande stormo. Ciò è dovuto al fatto che l'area della simulazione è molto maggiore di quella di influenza tra i singoli boids, per questo i vari piccoli stormi si uniscono solo quando si

avvicinano tra loro.

Alla creazione di un unico stormo, seguono una diminuzione della distanza media tra i boids e della deviazione standard della loro velocità, in quanto, ovviamente, le velocità dei singoli boids tendono a uniformarsi. Tale processo è fortemente legato sia ai dati forniti in input, sia dalla presenza o meno del predatore: lo scopo di quest'ultimo, infatti, non è solo quello di eliminare i boids, ma anche di disturbare la formazione dello stormo (nell'atto pratico disturba l'applicazione delle tre regole di volo).

## 5 Strategie di test

Una volta compilato il codice sorgente con CMake, viene restituito in output l'eseguibile `progetto/simulation.t` che ha il compito di testare le funzionalità implementate utilizzando il framework DOCTEST. Questa è una parte essenziale del progetto, in quanto consente di verificare il corretto funzionamento del programma e di testare il suo comportamento nei casi limite. Per esempio, la presenza di soli due boids rappresenta una situazione particolare, in quanto la deviazione standard della media delle distanze non assume più senso statistico (formalmente si avrebbe una divisione di una certa quantità per zero, che matematicamente, approssimando, risulterebbe  $\infty$ ); in tal caso, infatti, si è deciso di procedere mostrando la scritta *nan* ("Not-a-number") e non permettendo più al predatore di eliminare boids.

Come già anticipato, anche la presenza di un singolo boid risulta ambigua (si è scelto quindi di escludere questo caso al momento della scelta del numero di boids, attraverso un `runtime_error`), sia da un punto di vista fisico sia per come sono state implementate le funzioni di calcolo degli output.

In definitiva, i test implementati mirano a verificare il corretto comportamento del programma soprattutto in tali situazioni.

È importante testare il programma in condizioni limite per cercare di trovare e correggere anche errori poco probabili. A tal fine, le funzioni sono state provate anche nei casi in cui i boids occupino la stessa posizione, siano ad una distanza uguale a quella di separazione o quella massima alla quale possano interagire gli uni con gli altri, abbiano velocità troppo elevate o troppo basse. Per verificare il comportamento dell'evoluzione dei boids ai bordi della simulazione (garantito dalla funzione `pacman`) si è scelto di posizionarne alcuni proprio ai limiti della finestra grafica.

Tutti i test effettuati forniscono un esito positivo, dando così la certezza del corretto comportamento delle funzioni utilizzate e, di conseguenza, di tutto il programma.

## 6 Conclusioni

Il programma non presenta errori durante la compilazione, lo stormo si muove in modo coeso e tutte le regole di volo vengono rispettate. Inoltre, l'implementazione del predatore permette di spostare e "spezzare" lo stormo utilizzando la tastiera. La possibilità di modificare il numero di boid e di fermare l'evoluzione dello stormo aiuta a comprendere il comportamento del singolo boid rispetto a tutti gli altri. Manipolando i parametri delle regole di volo, durante la simulazione è possibile osservare come, aumentando i valori di  $s$ ,  $a$ ,  $c$ , lo stormo tenda rispettivamente a disperdersi di più, ad assumere una conformazione sempre più uniforme e a raggrupparsi quasi a delineare una forma geometrica circolare.

Interpretando tali risultati si ha un'ulteriore conferma della correttezza della simulazione, infatti a una separazione maggiore corrisponde maggiore dispersione, incrementando l'allineamento è ragionevole pensare che ogni boid dello stormo tende ad assumere una direzione del moto più in linea con gli altri ed aumentando la coesione i boids sono "attratti" dal centro di massa dei loro vicini. Oltre a ciò, attraverso l'aggiornamento di distanza e velocità media, è possibile capire meglio ciò che avviene durante l'evoluzione da un punto di vista più quantitativo.

## 7 Modifiche intercorse dalla consegna del progetto precedente

Il progetto si differenzia da quello precedentemente consegnato praticamente in ogni file. Il cambiamento più importante coinvolge l'uso degli algoritmi, scelti come alternativa più efficiente e chiara rispetto ai cicli `for` o `while`.

Altre modifiche afferiscono ai file "functions", specialmente per l'implementazione o l'aggiunta delle funzioni legate al calcolo di distanza, velocità e rispettive deviazioni standard, così come all'aggiunta di test.

Si è reso necessario anche definire una velocità minima dei boids nei file "evolve", al fine di evitare la possibilità che questi ultimi rallentino fino a (quasi) fermarsi, comportamento poco naturale per uno stormo.

Un altro cambiamento interessa il controllo degli errori, sia in fase di scrittura di codice (sono stati aggiunti degli assert per verificare la consapevolezza e la correttezza del codice che si stava scrivendo) che in fase di input. Nello specifico, nel "main" si è cercato di prevenire l'immissione di valori ambigui o non accettati dal programma:



1. permettendo all'utente di riprovare l'inserimento qualora venissero commessi errori di digitazione;
2. nel peggiore dei casi, se vengono inseriti valori non accettabili (come ad esempio l'inserimento di un numero di boids minore di due), il programma viene immediatamente terminato, evidenziando la motivazione del fallimento.

Le ultime modifiche interessano la parte grafica, con l'inserimento di uno sfondo più realistico per la simulazione (l'immagine importata si trova nel file `progetto/clouds.png`) e l'aggiunta di alcune configurazioni di CMake presenti nel file `progetto/.vscode/settings.json`.