

# Project PAXO

Andrea Barbieri, Michele Folli, Caterina Gasparrini, Leonardo Aielli

30 agosto 2024

[Link to repository](#)

## 1 Introduzione

Project PAXO ha l'obiettivo di simulare il comportamento di **BOIDS** (Bird-oid Objects) immaginabili come uno stormo di uccelli in volo in uno spazio bidimensionale, ispirandosi al [modello originale](#) descritto nell'articolo di Craig Reynolds datato 1987. Gli oggetti della simulazione sono rappresentati da *boids*, con due coordinate,  $x$  e  $y$ , di posizione e di velocità. I *boids* interagiscono nello spazio tramite tre regole di volo: separazione, allineamento e coesione, il cui obiettivo complessivo è creare uno stormo unico, coeso e dove le velocità dei singoli sono allineate.

Le tre regole si applicano solo ai *boids* abbastanza vicini, ovvero, dato un *boi*  $b_i$ , i *boids*  $b_j$  per cui:

$$\|\vec{x}_{b_i} - \vec{x}_{b_j}\| < d$$

dove  $d$  rappresenta il raggio visivo dei boid. Le regole di volo contribuiscono a modificare la velocità del *boi*, che alla fine dell'aggiornamento risulterà:

$$\vec{v}_{b_i} = \vec{v}_{b_i} + \vec{v}_s + \vec{v}_a + \vec{v}_c$$

dove  $\vec{v}_s$ ,  $\vec{v}_a$  e  $\vec{v}_c$  sono le velocità calcolate secondo le tre regole. La posizione finale invece sarà:

$$\vec{x}_{b_i} = \vec{x}_{b_i} + \vec{v}_{b_i} \Delta t$$

In particolare, le tre regole di volo sono la *separazione*, l'*allineamento* e la *coesione*. La prima allontana il *boi* dai vicini e impedisce che i *boids* collidano tra loro:

$$\vec{v}_s = -s \sum_{j \neq i} (\vec{x}_{b_j} - \vec{x}_{b_i}) \quad \text{se } \|\vec{x}_{b_i} - \vec{x}_{b_j}\| < d_s$$

dove  $d_s$  è una distanza minore del raggio di vista dei *boids* ( $d$ ) che stabilisce quando viene applicata la separazione, mentre  $s$  è il fattore di separazione.

La seconda regola tende ad allineare i *boids* tra loro, facendo procedere ognuno nella direzione dello stormo:

$$\vec{v}_a = a \left( \frac{\sum_{j \neq i} \vec{v}_{b_j}}{n-1} - \vec{v}_{b_i} \right)$$

dove  $a < 1$  è il fattore di allineamento e  $n$  è il numero di *boids* nel raggio visivo di *boi*  $b_i$ , compreso *boi*  $b_i$ .

La terza regola, infine, indirizza il *boi* verso il baricentro di quelli vicini, dato da:

$$\vec{x}_c = \frac{1}{n-1} \sum_{j \neq i} \vec{x}_{b_j}$$

con  $n$  come prima. Dato  $c$  fattore di coesione:

$$\vec{v}_c = c(\vec{x}_c - \vec{x}_{b_i})$$

A differenza del modello originale, nella versione implementata, i *boids* non interagiscono solo tra loro, ma anche con i bordi: quando un *boi* si avvicina troppo al bordo, gli viene applicata una

sorta di forza repulsiva che lo allontana, cambiando la sua velocità. Questo fa in modo che nessun *boïd* possa uscire dai confini della simulazione. Il meccanismo repulsivo ai bordi segue la seguente legge:

$$\begin{cases} v'_x = v_x \pm \|\vec{v}_{b_i}\| \left( \frac{1}{d_{bordo_x}} \right)^{\frac{1}{4}} & \text{se } d_{bordo_x} < d_{min} \\ v'_y = v_y \pm \|\vec{v}_{b_i}\| \left( \frac{1}{d_{bordo_y}} \right)^{\frac{1}{4}} & \text{se } d_{bordo_y} < d_{min} \end{cases}$$

Dove  $v'_x$  e  $v'_y$  sono le velocità aggiornate e  $d_{bordo_x}$  e  $d_{bordo_y}$  sono le distanze dalle rispettive pareti. Dato che ci sono due bordi orizzontali e due verticali, sia i segni degli addendi correttivi, sia le scelte delle pareti rispetto a cui  $d_{bordo_x}$  e  $d_{bordo_y}$  sono calcolate, vengono valutati appropriatamente ad ogni applicazione. Sia la la distanza massima di interazione  $d_{min}$  che l'esponente 0.25 sono stati scelti arbitrariamente.

Un'altra aggiunta che è stata fatta al modello è stata l'imposizione di una velocità limite scelta con un criterio puramente grafico.

Le posizioni e le velocità iniziali dei *boïds* vengono generate casualmente, poi questi iniziano a interagire tra di loro in base alle regole di volo e con i bordi e dopo un certo tempo dovrebbero formare uno stormo coeso, quindi con piccole distanze tra i singoli *boïds*, e con velocità allineate tra loro. Per verificare che questo accada, durante la simulazione vengono raccolte una serie di statistiche, quali la velocità e la distanza media dello stormo e le loro deviazioni standard.

## 2 Design e implementazione

### 2.1 Struttura programma

Il progetto è composto da 5 files: **Boids.hpp**, **Boids.cpp**, **IOhandling.hpp**, **IOhandling.cpp** e **main.cpp**. In **Boids.hpp** sono definiti tutti gli *user defined types* e sono dichiarate tutte le rispettive funzioni membro insieme alle funzioni che gestiscono la simulazione e il calcolo delle statistiche, mentre in **IOhandling.hpp** sono dichiarate tutte le funzioni che gestiscono i processi di input e output. Le definizioni di tutti gli operatori e di tutte le funzioni sono invece date nei rispettivi files **.cpp**; il file **main.cpp** è invece riservato alla funzione main. Si era preso in considerazione di suddividere ulteriormente il progetto rimuovendo la struct **Stats** e le funzioni statistiche dai file **Boids** e dedicandogli un *header* e un *.cpp* proprio. Questa opzione è stata scartata dato il limitato numero di funzioni, oltre alla loro stretta correlazione con gli oggetti *Boid*. Nonostante ciò, se in futuro la componente statistica del progetto dovesse essere ampliata, questa separazione verrebbe sicuramente implementata.

Il programma si serve inoltre delle librerie esterne **SFML** e **Gnuplot**: la prima è usata per il comparto grafico della simulazione, mentre la seconda per la creazione dei grafici delle statistiche.

Gli *user defined types* introdotti sono 4: *Vec\_2d*, *Params*, *Stats* e *Boid*. I primi 3 sono structs mentre l'ultimo è definito tramite una class. *Stats* e *Params* hanno la sola funzione di unire 5 floats ciascuno, rispettivamente le statistiche e i parametri della simulazione, senza particolari funzionalità. Al contrario la class *Boid* è particolarmente articolata dovendo gestire i principali oggetti della simulazione, mentre *Vec\_2d*, struct di supporto alla class, è dotata solo delle principali operazioni algebriche dei vettori. Nello specifico, la class ha come componenti private 2 variabili membro *position* e *velocity* di tipo *Vec\_2d* che descrivono le proprietà fisiche dei *boïds* e 4 funzioni membro che descrivono tutte le regole di volo; queste funzioni vengono riassunte nel membro pubblico *update* che aggiorna le coordinate del *boïd* su cui viene chiamato. Le altre funzioni membro forniscono uno strato di utility per le funzioni esterne alla classe.

Nel complesso, le funzioni del progetto gestiscono i 4 aspetti principali dell'esecuzione: input, esecuzione della simulazione, statistiche e output. Nelle fasi di input e output il programma si propone di essere il più elastico possibile. Le rispettive funzioni forniscono molte opzioni di utilizzo con funzionalità quali: input di vari parametri, stampa di statistiche con annessa creazione di grafici con l'ulteriore opzione di salvare questi ultimi sotto forma di file **.txt** e **.png**. Durante la simulazione le due principali funzioni che operano sono *runSimulation* e *updateStats*: la prima si occupa di aggiornare e disegnare lo stormo di *boïds*, composto da un *std::vector<Boid>*, mentre la seconda legge periodicamente il vettore per calcolare e accumulare le statistiche. Entrambe queste funzioni iniziano ad operare all'apertura della finestra in cui è rappresentata la simulazione e continuano fino alla sua chiusura, valutando ad ogni iterazione la condizione "*window.isOpen()*", fornita da SFML.

Per far sì che la lettura del vettore e il calcolo delle statistiche possano avvenire senza bloccare il processo di simulazione si è scelto un approccio che fa uso del multithreading. Dopo l'inizializzazione di tutti i suoi oggetti la funzione `main` costruisce un thread, nel quale viene chiamata la funzione `updateStats`, che opera in parallelo al main thread. Nel thread principale viene invece chiamata `runSimulation`. Entrambe le funzioni operano fino alla chiusura della finestra, dopo la quale il thread parallelo viene ricongiunto a quello principale.

Per assicurarsi che durante il calcolo delle statistiche venga letta una versione dello stormo che non sta venendo modificata, ci si serve di `std::mutex`, in modo che il vettore non possa venir letto durante la modifica e non possa essere modificato durante la lettura. Data l'implementazione di `std::mutex`, un thread non può accedere ad un oggetto che sta venendo modificato o visionato da un altro thread. Questa attesa forzata può portare ad un arresto temporaneo della simulazione la quale, per aggiornare il vettore che rappresenta lo stormo, deve attendere che sia terminato il calcolo delle statistiche. Per ridurre al minimo questa finestra, nella funzione `main` sono definiti 2 `std::vector<Boid>`: **`flock`** e **`flock_view`** che vengono poi passati by reference alle due funzioni. In questo modo la funzione `runSimulation` aggiorna il vettore `flock` e al termine dell'aggiornamento assegna i nuovi valori a `flock_view`, che viene poi letto per calcolare le statistiche. Questa soluzione restringe il più possibile la finestra di condivisione tra i due thread, riduce drasticamente il lag e fa sì che venga sempre letta una versione "consistent" dello stormo nel calcolo delle statistiche.

## 2.2 Implementazione modello BOID

La simulazione si svolge in un ambiente limitato, ovvero la finestra stessa, perciò ci si prefigge di simulare l'interazione dello stormo entro questi confini. Un'interpretazione fisica di questa scelta è data dal concetto di velocità relativa. Pensando ad uno stormo di uccelli migratori, è facile immaginare come essi, pur condividendo una velocità indirizzata verso la medesima direzione, si muovano anche l'uno relativamente all'altro. La finestra si può quindi vedere come un sistema di riferimento in moto rettilineo uniforme nel quale il comportamento dei *boids* descrive l'evoluzione delle velocità relative di uno stormo di uccelli. La simulazione si prefigge quindi di studiare l'evoluzione dello stormo sotto queste ipotesi.

Dovendo far rimanere tutti i *boids* nei confini della finestra era necessaria l'implementazione di un meccanismo ai bordi che interferisse il meno possibile con il comportamento dello stormo in modo da garantire una simulazione il più possibile fedele al modello originale. Si è optato quindi per il modello descritto precedentemente nella sez. [1]. Un'altra opzione presa in considerazione era l'implementazione di un modello toroidale che avrebbe portato i *boids* usciti da un bordo a ricomparire dal lato opposto, analogamente al cosiddetto "effetto Pac-man". Questa opzione è stata scartata in quanto interferiva fortemente con l'abilità dello stormo di rimanere coeso. Nonostante sia molto meno invasivo del modello toroidale, anche il modello implementato con pareti repulsive ha degli effetti non trascurabili sullo stormo. Infatti, per quanto l'effettiva coesione dello stormo non sia influenzata, le velocità risentono molto questa influenza. Questo perché lo stormo che si dirige verso la parete, una volta raggiunta una distanza critica, "rimbalza" indietro, causando quindi fluttuazioni nelle velocità in concomitanza dell'urto dello stormo con la parete. Questi effetti verranno analizzati nella sezione successiva.

Nel suo complesso il modello che descrive lo stormo è stato implementato aggiornando ad ogni frame della simulazione velocità e posizione di tutti i boid dello stormo tramite la funzione `update`. Questa infatti modifica le velocità di ogni *boid* applicando: le 3 regole di volo, il metodo `avoidEdges`, che tiene conto delle pareti repulsive, e il metodo `limit` per limitare la velocità dei *boids*. La velocità limite è di 10 pixel/s ed è stata scelta in maniera arbitraria. La posizione viene poi aggiornata sommando ad essa il vettore velocità appena calcolato: si considera infatti  $\Delta t = 1s$ . Questo significa che ad ogni aggiornamento del vettore, corrispondente ad un frame di simulazione, il tempo trascorso nell'universo della simulazione è di 1s.

Si è scelto inoltre di adottare un sistema di aggiornamento "preferenziale" nei confronti dei primi *boids* del vettore. Infatti, per come è strutturato il loop di aggiornamento, ogni *boid* viene aggiornato in base alla sua posizione nel vettore, a partire dal primo fino all'ultimo. Perciò, quando un *boid* viene aggiornato tiene conto dello stato già aggiornato dei *boids* a lui precedenti, seguendo gli indici del vettore, mentre i *boids* successivi sono ovviamente nello stato precedente. Questo sistema risulta preferenziale nei confronti dei primi *boids* del vettore in quanto questi tengono conto dello stato precedente di tutti gli altri, a differenza dei successivi che vedono versioni già aggiornate dei *boids* che li precedono e tendono quindi ad allinearvisi.

Questa implementazione era nata come errore, ma, una volta corretto, si è osservato come un modello che aggiorna i *boids* solo in funzione dello stato precedente del vettore tenda a produrre simulazioni estremamente statiche. Infatti in queste simulazioni si formava molto rapidamente

qualche stormo, il cui movimento macroscopico cessava dopo pochi secondi, impedendo la creazione di un unico stormo. La staticità delle configurazioni sembra dovuta all'apparente incapacità delle velocità dei singoli *boids* di allinearsi, causando quindi stormi "pulsanti" senza però alcun senso di direzione comune.

Queste osservazioni hanno portato alla teoria che un modello preferenziale verso i primi *boids* del vettore sia necessariamente più dinamico in quanto i primi elementi del vettore stormo sono meno influenzati dal resto dagli altri e fungono quasi da guide per il resto dello stormo. La teoria sembra trovare ulteriore conferma nel fatto che test svolti sul modello preferenziale, dove però l'ordine degli elementi del vettore veniva casualmente ridistribuito alla fine di ogni ciclo di update, per far sì che nessun *boid* fosse effettivamente privilegiato, producevano configurazioni statiche estremamente simili a quelle del modello non preferenziale. Si è quindi deciso di implementare il modello preferenziale al fine di ottenere simulazioni dinamiche che mettano effettivamente alla prova il modello.

## 2.3 Statistiche

L'analisi statistica ha l'obiettivo di verificare che il modello simuli correttamente il comportamento dello stormo. Si prevede che l'evoluzione dello stormo nel tempo porti ad uno stormo coeso le cui velocità sono allineate.

Per analizzare la coesione si sono usate la distanza media e la sua deviazione standard:

$$\bar{d} = \frac{1}{N} \sum_{i \neq j} d_{ij} \quad \sigma_d = \sqrt{\frac{1}{N-1} \sum_{i \neq j} (d_{ij} - \bar{d})^2} \quad N = \frac{n(n-1)}{2}$$

Dove  $d_{ij}$  rappresenta la distanza tra una generica coppia di *boids*,  $n$  rappresenta il numero totale di *boids* e  $N$  rappresenta il numero totale di distanze calcolabili, ovvero il numero di elementi della sommatoria.

Data la sua natura vettoriale, la trattazione della velocità è più complicata di quella della distanza, che essendo una grandezza scalare, si può trattare facilmente con metodi classici. È molto importante specificare che i termini **velocità media** e **deviazione standard della velocità** non sono pienamente indicativi dell'implementazione effettiva a cui fanno riferimento; nonostante ciò sono stati comunque adottati per semplicità e per motivi storici nell'evoluzione del progetto.

Allo scopo di valutare l'allineamento delle velocità si è deciso di calcolare il vettore medio e di trovarne la norma con un approccio analogo alla [mean resultant length](#), a differenza di quest'ultima però si è scelto di non utilizzare vettori normalizzati. La velocità media risulta quindi:

$$\bar{\mathbf{v}} = \|\langle \vec{v} \rangle\| \quad \text{dove : } \langle \vec{v} \rangle = \sum_{i=1}^n \frac{\vec{v}_i}{n}$$

Per quanto riguarda la deviazione standard si è invece deciso di non usare altri elementi della statistica circolare. Infatti nessuna misura della varianza circolare sembrava aggiungere un numero significativo di informazioni. Si è quindi implementata la seguente deviazione standard con l'obiettivo di quantificare lo scarto quadratico medio tra un generico vettore velocità e il vettore velocità medio:

$$\sigma_v = \sqrt{\sum_{i=1}^n \frac{\|\vec{v}_i - \langle \vec{v} \rangle\|^2}{n-1}}$$

Nonostante le implementazioni della velocità media e della deviazione standard non siano apparentemente collegate ci si aspetta comunque un collegamento tra l'andamento nel tempo delle due grandezze. Infatti, dato che un aumento nella velocità media è dovuto ad un crescente allineamento, è ragionevole aspettarsi che questo porti anche ad una riduzione della deviazione standard.

## 3 Strategia di test

Il programma è stato testato tramite [doctest](#) per verificare che le funzioni si comportassero come voluto. In particolare sono state testate la funzione che calcola la distanza assoluta tra due *boids*,

le tre regole di volo sia separatamente sia tutte insieme e poi insieme alla funzione che gestisce l'interazione con i bordi e quella che limita la velocità, le quali sono state testate anche singolarmente. Infine sono state testate anche le statistiche. Tutti i test sono contenuti nel file **Boidtest.cpp**. Per verificare, invece, che il comportamento di stormi di molti *boids* fosse effettivamente fedele alle previsioni del modello, si sono osservate sia la simulazione grafica, sia le statistiche mostrate in output. Per un'analisi più approfondita dei risultati di queste osservazioni rimandiamo alla sez.[5.2].

## 4 Come utilizzare il programma

### 4.1 Comandi per compilare, testare ed eseguire

Perché il programma possa operare ed essere compilato correttamente sono necessari: la libreria grafica SFML, la libreria Gnuplot e il Build System CMake. Per installarli è sufficiente eseguire il comando:

- **\$ sudo apt update && sudo apt install cmake libsFML-dev gnuplot**

Per scaricare la repository del progetto da Github:

- **\$ git clone https://github.com/Michelefolli/Progettopaxxo.git**

Per la compilazione è necessario eseguire questi due comandi a partire dalla cartella del progetto:

- **\$ cmake -S ./Source/ -B release/ -DCMAKE\_BUILD\_TYPE=Release**
- **\$ cmake --build release/**

Partendo dalla stessa cartella, per il corretto caricamento degli asset, l'esecuzione del programma dev'essere fatta col seguente comando:

- **\$ release/Project\_PAXO**

Per eseguire i test:

- **\$ cmake --build release/ --target test**

Alternativamente si può anche eseguire l'eseguibile contenente i test col comando:

- **\$ ./release/Boid\_test**

### 4.2 Input

Quando si esegue il programma compare l'indicazione *"Input the required data:"* in questo modo l'utente del programma può inserire in input il numero di *boids* che vuole nella simulazione, il periodo di acquisizione delle statistiche (misurato in millisecondi) e i parametri delle regole di volo, come richiesto passo passo dalle indicazioni che compaiono sulla shell. Il programma si assicura che gli input inseriti siano validi e permette di inserire nuovamente gli input giudicati invalidi. Riportiamo i range di validità per i valori in input:

- *numero di boids* > 1
- *periodo di acquisizione* > 10
- *fattore di separazione* > 0
- $0 < \text{fattore di allineamento} < 1$
- *fattore di coesione* > 0
- $0 < \text{raggio di applicazione della regola di separazione}$
- *raggio di applicazione della regola di separazione* < *raggio visivo dei boids*

**N.B.:** È stato scelto il valore di 10ms come minimo per il periodo di acquisizione delle statistiche in funzione delle 2 cifre decimali fornite in output dal programma per quanto riguarda la statistica tempo; nonostante ciò è fortemente sconsigliato utilizzare valori così piccoli per motivi di valenza delle statistiche. Il programma infatti segue la prestazione massima del dispositivo su cui opera, prova quindi ad aggiornare lo stormo con una frequenza uguale al refresh rate dello schermo del dispositivo, velocità di calcolo permettendo. Questo fa sì che periodi di acquisizione delle statistiche inferiori al reciproco del refresh rate dello schermo portino a molteplici acquisizioni delle statistiche a partire dallo stesso set di dati. Si consiglia quindi di non utilizzare valori troppo bassi per la grandezza in questione. Le seguenti linee guida danno quindi un riferimento per i valori minimi consigliati, che sono comunque da implementare in funzione delle prestazioni della macchina e del refresh rate della macchina:

- *periodo di acquisizione* > 50 per computer portatili non collegati alla corrente
- *periodo di acquisizione* > 20 per computer fissi e portatili collegati alla corrente

Dispositivi particolarmente potenti dotati di schermi ad alte prestazioni possono sperimentare anche con valori del periodo particolarmente bassi.

### 4.3 Linee guida per l'utilizzo

Vengono forniti di seguito alcuni consigli sui valori da assegnare ai parametri per ottenere delle simulazioni grafiche significative. In linea generale, aumentando il fattore di separazione si ottengono *boids* meno ravvicinati, quindi stormi più caotici; aumentando l'allineamento le velocità tendono ad allinarsi maggiormente quindi i *boids* vanno più veloci e aumentando la coesione, si ottengono appunto stormi più coesi. Aumentando il raggio visivo, invece, i *boids* si uniscono più facilmente, in quanto aumenta l'interazione tra gli stormi vicini, mentre aumentando la distanza di separazione, otteniamo un effetto simile all'aumento del fattore di separazione, quindi comportamento caotico e *boids* che si respingono.

Si riportano ora dei set di parametri che possono essere presi in considerazione dall'utente per iniziare a familiarizzare con il programma.

- **Preset 1:**

Il primo set proposto vede i seguenti parametri **{300, 0.7, 0.9, 0.3, 60, 20}**, nell'ordine: numero di *boids*, fattore di separazione, fattore di allineamento, fattore di coesione, raggio visivo, distanza di separazione. Con questi valori si ottiene una simulazione molto fluida e dinamica, nella quale i *boids* diventano relativamente in fretta un unico stormo, non eccessivamente coeso, ma molto dinamico. Può capitare che dopo che si è formato un unico stormo questo si separi, per poi unirsi nuovamente. Aumentando il numero di *boids*, per esempio a 600, la velocità dei *boids* diminuisce e aumenta il rischio per lo stormo già unito di dividersi.

- **Preset 2:**

Se l'utente desidera uno stormo più coeso e con meno rischio di divisione, si propone questo set: **{600, 0.5, 0.7, 0.5, 70, 25}**. Diminuendo la separazione e aumentando la coesione, inevitabilmente il movimento diventa meno fluido, ma una volta che si è creato lo stormo unico, è molto difficile che si divida. Per poter distinguere chiaramente i *boids* si è dovuta aumentare la distanza di separazione, che in ogni caso, indipendentemente dai parametri, si sconsiglia di portare al di sotto di 15, per evitare che i *boids* si sovrappongano. L'aumento della coesione, invece, obbliga a un aumento del raggio visivo per facilitare la creazione dello stormo unico, che però perde di dinamicità e risulta un po' statico nella configurazione finale.

- **Preset 3:**

Un ottimo set per uno stormo coeso, ma dinamico è: **{600, 0.35, 0.8, 0.4, 60, 20}**. Diminuire il fattore di separazione permette di diminuire leggermente anche la coesione, ottenendo comunque uno stormo compatto; questo rende possibile ridurre anche il raggio visivo. Con questi valori si ottiene uno stormo coeso, che difficilmente si separa dopo essersi unito, ma che rimane ugualmente molto dinamico, grazie anche all'aumento dell'allineamento.

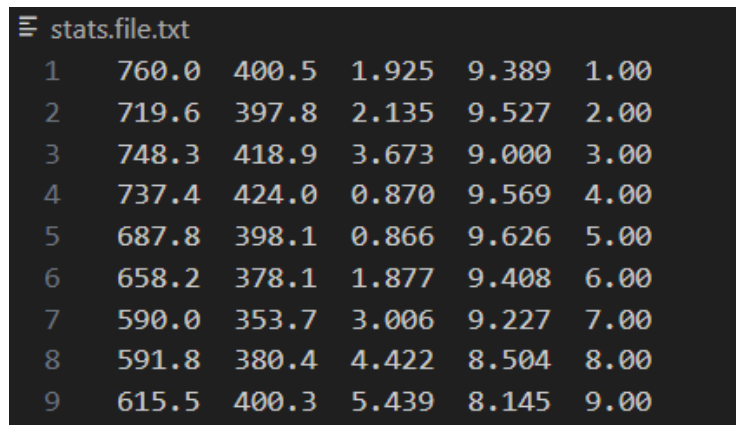
- **Preset 4:**

Se quello che l'utente cerca è, invece, uno stormo più libero e meno coeso, non è sufficiente alzare il fattore di separazione: questo infatti costringe ad aumentare la coesione per poter creare un unico stormo, con perdita di dinamicità. Si consiglia, invece, di abbassare la

coesione e contemporaneamente di alzare, solo leggermente, la separazione. Con il set **{400, 0.6, 0.9, 0.2, 60, 20}** si ottiene una simulazione molto dinamica e meno coesa, ma nella quale riesce comunque a crearsi un unico stormo, anche se soggetto al rischio di sfaldarsi. Con questo livello di libertà e con un fattore di allineamento così alto, i *boids* sono molto veloci, quindi si sconsiglia di mettere un numero di *boids* troppo basso.

Queste linee guida forniscono un buon punto di partenza per capire come la variazione dei parametri influenzi la simulazione. A questo punto l'utente può poi sperimentare a suo piacimento, ricordando che aumentare troppo il raggio visivo rende molto statico lo stormo, mentre ridurlo troppo rende difficile la creazione di un unico stormo, la quale può essere ottenuta con i giusti parametri, ma non è garantita a prescindere da essi.

## 5 Output del programma



	1	2	3	4	5
1	760.0	400.5	1.925	9.389	1.00
2	719.6	397.8	2.135	9.527	2.00
3	748.3	418.9	3.673	9.000	3.00
4	737.4	424.0	0.870	9.569	4.00
5	687.8	398.1	0.866	9.626	5.00
6	658.2	378.1	1.877	9.408	6.00
7	590.0	353.7	3.006	9.227	7.00
8	591.8	380.4	4.422	8.504	8.00
9	615.5	400.3	5.439	8.145	9.00

Figura 1:

Questo è un esempio dell'interfaccia del file `.txt` sul quale sono state esportate le statistiche. Il set di dati utilizzato per questa simulazione è stato il **Preset 3**

### 5.1 Dati in output

Una volta inseriti i valori richiesti, il programma inizializza l'interfaccia grafica della simulazione, implementata con SFML, dove si può visualizzare il movimento dei *boids*. Quando si chiude la finestra dell'interfaccia grafica, il programma chiede all'utente se vuole esportare le statistiche come file `.txt` (Fig.1): *"Input 1 if you want to export the stats as a .txt file, otherwise input 0"* e controlla sempre che l'input sia valido. A prescindere dalla decisione, vengono comunque stampate sulla shell le statistiche con lo stesso formato del file di testo per permetterne una rapida visualizzazione, divisi in cinque colonne, nell'ordine: distanza media, deviazione standard della distanza, velocità media, deviazione standard della velocità e istante di tempo di acquisizione. Se l'utente decide di esportare le statistiche viene creato il file e il programma dà anche la possibilità di nominarlo. All'interno del file le quattro colonne sono organizzate come sulla shell.

Il programma è in grado anche di creare dei grafici bidimensionali per ognuna delle quattro statistiche, che riportano il tempo in ascissa e la statistica in ordinata. Anche in questo caso il programma chiede all'utente se vuole esportare i grafici come file `.png`: *"Input 1 if you want to export the plots as a .png file, otherwise input 0"* e controlla la validità dell'input. Premendo 1 si ha la creazione del file `.png` che può essere nominato, se si preme 0, invece, si apre solo una finestra con i grafici, che una volta chiusa non rimane salvata.

### 5.2 Analisi degli output

L'output del programma è stato anche uno strumento utile per testare il codice e verificare che si comportasse come previsto dal modello. La simulazione grafica ha determinato alcune scelte di implementazione e confermato che il modo in cui si erano scritte le regole di volo producesse il comportamento desiderato. L'obiettivo, infatti, era fare in modo che la simulazione grafica fosse fluida e dinamica e che i *boids* non uscissero dai bordi. La simulazione ha anche dato una conferma



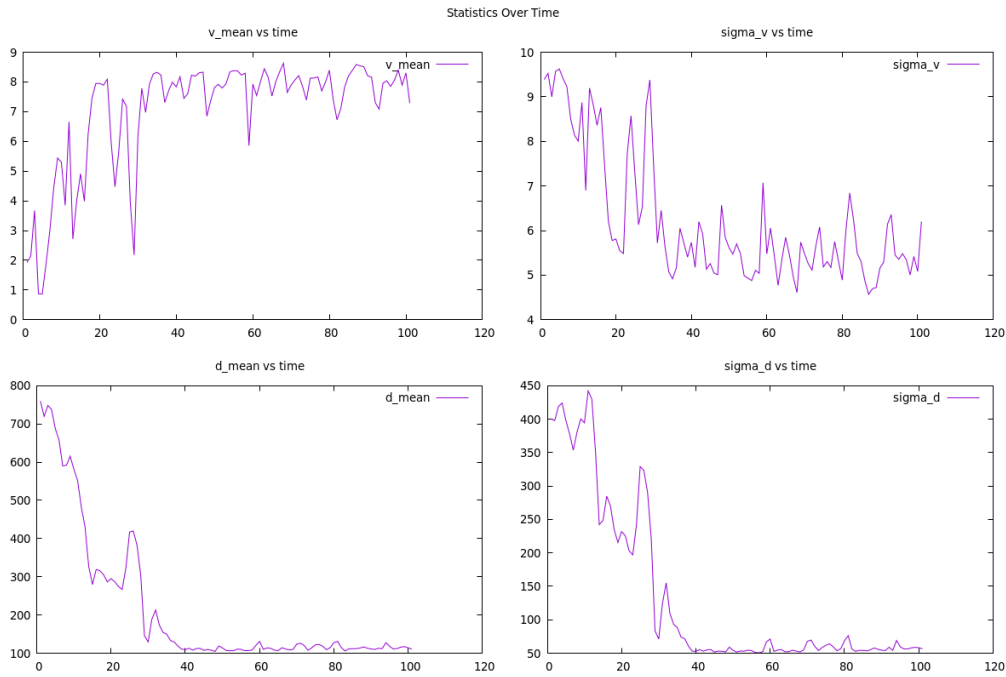


Figura 2:

Questo è un esempio di file .png sul quale sono state esportate le statistiche. Il set di dati utilizzato per questa simulazione è stato il **Preset 3**

preliminare che, dopo un certo tempo, si creasse effettivamente uno stormo coeso con velocità allineate. Scegliendo valori adeguati dei parametri, si può osservare che si ottiene effettivamente questo risultato.

Anche i grafici delle statistiche in funzione del tempo costituiscono una conferma significativa del comportamento lecito del codice. Osservandoli, infatti, si possono notare alcune caratteristiche che corrispondono a quello che ci si aspettava. È fornito un esempio in [Fig.2].

Il grafico in alto a sinistra è quello della velocità media, il cui andamento oscillatorio è dovuto all'interazione con i bordi. È interessante confrontare questo grafico con quello della sua deviazione standard: le due statistiche hanno andamento opposto. Infatti un aumento della velocità media è indice del crescente allineamento delle velocità dei boid, che si traduce in una riduzione della deviazione standard. Anche il viceversa è vero: riduzioni della velocità media sono sinonimi di un aumento della deviazione standard.

Osserviamo ora i grafici delle statistiche sulle distanze. Questi sono relativi a una simulazione fatta con 600 *boids*, che ha raggiunto la configurazione di coesione. Intorno ai 40s, infatti, vediamo che la distanza media si abbassa notevolmente e rimane approssimativamente costante. Anche per la distanza le oscillazioni sono dovute all'interazione dello stormo con le pareti. Al contrario delle statistiche sulla velocità, distanza media e deviazione standard della distanza presentano lo stesso andamento. Questo è dovuto al fatto che una riduzione della distanza media è segnale di una configurazione coesa, dove quindi anche lo scarto quadratico medio delle distanze diminuisce.

Tutti questi dati confermano le previsioni e sono coerenti con la configurazione dello stormo ricercata.

### 5.3 Errori librerie esterne

L'utilizzo di librerie esterne può portare a qualche errore al termine dell'esecuzione, mostriamo quindi come risolverli.

- **"QStandardPaths: wrong permissions on runtime directory":**

L'utilizzo della libreria **Gnuplot** porta alla presenza di errori del tipo:

**"QStandardPaths: wrong permissions on runtime directory /run/user/1000/, 0755 instead of 0700"**



Questo errore è causato dall'utilizzo che Gnuplot fa del **framework Qt**, strumento alla base di molte librerie grafiche. Infatti è dovuto al fatto che i permessi per la cartella `/run/user/1000/` non sono settati a 0700. Per risolvere l'errore è quindi necessario cambiare i permessi con il seguente comando:

```
"$ chmod 0700 /run/user/1000/"
```

In caso fosse poi necessario per l'utente ritornare allo stato precedente della cartella sarà sufficiente rieseguire lo stesso comando, indicando però il livello di permessi iniziale. Nel caso dell'esempio proposto questo vorrebbe dire eseguire un comando del tipo:

```
"$ chmod 0755 /run/user/1000/"
```

La risoluzione di quest'errore non sembra comunque necessaria al fine di una corretta esecuzione del programma.

- **Memory leaks**

Se si compila il programma servendosi di impostazioni che monitorano il memory leak, come *AddressSanitizer*, è possibile osservare numerosissimi errori di questo tipo, la cui somma si aggira attorno ai  $\sim 130/140\text{kB}$ . La quantità "leakata" sembra non dipendere nè dal numero di boids nè dalla durata della simulazione; questo ha portato a concludere che l'errore sia dovuto agli elementi caricati da SFML. Questa teoria è stata confermata creando una versione provvisoria del codice che non si serve della libreria: l'eseguibile così prodotto, infatti, non ha mostrato alcun memory leak. Facendo ricerche sull'argomento, sono state trovate altre menzioni di errori simili, pare infatti che SFML "pulisca" gli oggetti utilizzati in un momento successivo alla scansione di *AddressSanitizer*, causando quindi gli errori sopracitati. I creatori di SFML sostengono che la libreria non produca errori di questo tipo, nonostante frequenti segnalazioni. Probabilmente quindi la spiegazione risiede nell'interazione tra la gestione degli oggetti della libreria esterna e l'operazione di monitoraggio di *AddressSanitizer*, senza portare però alcun problema nell'esecuzione del programma.