Simulazione del Comportamento di Stormi

Autori del progetto:

Milazzo Leandro, Milia Raffaele, Morselli Alessandro, Oliverio Luca

Data ultima modifica:

24/08/2025

Sommario

Modifiche effettuate dalla consegna precedente

Rispetto all'ultima versione sono state effettuate le seguenti modifiche:

- E' stata cambiata l'organizzazione dei boids: precedentemente era suddivisa in tre diverse struct, ovvero Boids, Boids_vel e Boids_Complete; nell'attuale versione la leggibilità del codice è stata migliorata creando un'unica struct Boid al cui interno vi sono due array di double a due dimensioni, uno per la posizione e uno per la velocità. Si è scelto di unificare posizione e velocità in un'unica struct Boid perchè rende il codice più compatto, più semplice da mantenere e migliora anche l'efficienza, perché i dati relativi a un singolo boid sono vicini in memoria. Bisogna però riconoscere che c'è una perdita di chiarezza semantica (Position e Velocity adesso hanno lo stesso tipo);
- La suddivisione del metodo update e del ciclo while nel main in più funzioni/metodi brevi ha migliorato la leggibilità e la manutenibilità del codice, rendendo più chiaro il flusso e favorendo i test e il riutilizzo di singoli blocchi di codice;
- Gestione più efficiente delle variabili, con riduzione dello scope quando possibile e applicazione dei qualificatori appropriati;
- Diversa gestione della generazione di numeri casuali;
- Implementazioni di nuovi metodi per rendere il codice più originale e dinamico. Sono quindi state introdotte nuove possibilità di interazione con l'utente, ovvero la possibilità di generare o rimuovere boids a runtime;

- Introduzione del fattore dt per uniformare il comportamento dei boid indipendentemente dagli FPS, evitando che a frame rate più elevati il movimento risulti amplificato e garantendo un'esecuzione uniforme nel tempo;
- Aggiunta di un contatore per stampare sul terminale le statistiche riguardanti il comportamento dei boids soltanto una volta al secondo (invece che ad ogni frame);
- Rivisitazioni generali delle funzioni, puntando a migliorarne l'efficienza e la leggibilità.

Tema del progetto e introduzione

Il progetto ha lo scopo di simulare sistemi complessi modellando il comportamento collettivo di entità autonome, ossia i boids. Tale modello riproduce la dinamica e la cinematica di stormi di uccelli, banchi di pesci o sciami di insetti, tramite la regolazione del moto di ogni boid sulla base di regole locali, quali i fattori di separazione, allineamento e coesione.

Il programma è stato suddiviso in 4 diversi file, ovvero:

- Boids_logic.hpp è l'header contenente tutte le definizioni delle classi/struct, le dichiarazioni dei metodi riguardanti la logica del sistema e la definizione di due funzioni inline. Inoltre presenta le dichiarazioni di due metodi per l'implementazione grafica (draw_mouse e draw_boids);
- Boids_logic.cpp, nel quale sono state definiti i metodi dichiarati precedentemente nell'hpp;
- Boids.test.cpp in cui vengono effettuati i test sulle funzioni implementate in precedenza;
- Main.cpp è il file che gestisce l'input dei parametri e contiene il ciclo for che, ad ogni frame, aggiorna i boid e ne rinnova la rappresentazione grafica.

Nell'hpp è stata definita la struct Boid, con due std::array<double, 2> che rappresentano posizione e velocità; per migliorarne la leggibilità sono stati introdotti degli alias tramite l'utilizzo di using. La classe Movement raccoglie invece i metodi relativi al moto dei boid e alla loro rappresentazione grafica, includendo le tre regole fondamentali (rule1, rule2 e rule3) che, sulla base dei parametri forniti, aggiornano la velocità di ciascun boid secondo i principi di separazione, allineamento e coesione.

Il metodo principale update(…) applica le regole tra i vari boids e ne aggiorna le posizioni. Su questi file sono stati eseguiti dei test per verificare il corretto funzionamento della logica del programma nel file "boids.test.cpp" utilizzando "doctest.h". Nel file main.cpp vengono richiesti in input i parametri del programma all'utente e si verifica che essi siano conformi alle regole definite utilizzando dei throw e dei catch.

Inoltre, la grafica è stata gestita tramite l'utilizzo della libreria SFML (*Simple and Fast Multimedia Library*), che consente la visualizzazione animata e l'interazione con la simulazione dei boids. In particolare, viene creata a runtime una finestra grafica 1600×900 pixel dove viene disegnata la simulazione, con limitazione del framerate impostata a 90 FPS ma facilmente modificabile nel main. La finestra viene modificata ad ogni frame tramite un ciclo for che aggiorna le posizioni e le velocità dei boids e gestisce le interazioni con l'utente a runtime. Vi è infatti la possibilità di generare nuovi boids durante l'esecuzione del programma premendo il tasto "Space" o di rimuoverli attraverso il tasto "R". Essi vengono rappresentati da puntini bianchi/rossi inizializzati con posizioni e velocità casuali, invece attorno al cursore viene visualizzato un cerchio colorato che rappresenta la zona

d'influenza entro cui i boids risentono della forza esercitata dal puntatore. Se il tasto sinistro non è premuto, la forza è attrattiva e il cerchio è verde; se, invece, il tasto sinistro è premuto costantemente, la forza diventa repulsiva e il cerchio rosso. Questa interazione è realizzata mediante il metodo set_mouse_force(...) della classe Movement, che aggiorna i parametri interni relativi alla posizione del mouse, allo stato della pressione e alla modalità della forza. Il comportamento fisico viene poi gestito nel metodo update(...), in cui la forza del mouse viene applicata all'interno di un raggio prefissato tramite il metodo privato apply_mouse_force(...).

Librerie esterne da installare

Abbiamo utilizzato due librerie esterne, ovvero:

- SFML per l'interfaccia grafica;
- CMake per la compilazione.

Istruzioni per eseguire il programma

Per eseguire il programma si sfrutta il sistema di build CMake. Descriviamo i passaggi per ottenere due eseguibili, che avranno il nome di "boids_sim" e "boids_sim.t", rispettivamente per la rappresentazione grafica con in input i parametri per la gestione delle regole del volo e per la verifica dei test inseriti nel file "boids.test.cpp" sfruttando "doctest.h". All'interno della cartella *Progetto_RLLA* si esegue sul terminale il comando:

cmake -S . -B build -G"Ninja Multi-Config"

che genera la directory *build* contenente tutto il necessario per la compilazione e la creazione degli eseguibili. Sempre da *Progetto_RLLA*, il passo successivo è:

cmake --build build --config Debug

che crea i due eseguibili in modalità *Debug*. Sostituendo *Debug* con *Release* si ottiene invece la versione ottimizzata. Infine, per avviarli, si utilizzano i comandi (in base alla configurazione scelta):

build/Debug/boids_sim
build/Release/boids_sim.t

Parametri di input e formato output

All'avvio della simulazione il programma richiede all'utente i parametri da inserire per la regolazione del comportamento cinematico e dinamico del sistema. I parametri richiesti sono elencati in seguito e sono accompagnati da un ipotetico valore standard da potergli assegnare.

Input da terminale:

- Numero di boids "n_b" (es. 200): è un intero positivo che definisce il numero di boids presenti nel sistema;
- Distanza di interazione "d" (es. 100): è un numero positivo che definisce la distanza per la quale un boids inizia a risentire delle interazioni dovute alla presenza degli altri boids;

- Distanza di separazione “d_s” (es. 20): è un numero positivo che definisce la distanza massima di avvicinamento tra due boids prima che si attivi la regola di separazione;
- Coefficiente di separazione “s” (es. 1.5): regola l’intensità della forza repulsiva;
- allineamento “a” (es. 0.04): regola l’intensità della forza di allineamento;
- coesione “c” (es. 0.3): regola l’intensità dell’attrazione dei boids verso il centro di massa;

Il programma tornerà in output:

- Finestra 1600×900 pixel con animazione dei boids;
- Boids disegnati come cerchi bianchi che tendono a diventare gradualmente più rossi all’aumentare della velocità;
- Cerchio interattivo attorno al mouse se la forza è attiva (verde per attrattiva, rosso per repulsiva);
- Statistiche generali del sistema stampate sul terminale una volta al secondo.

Interpretazione dei risultati

La simulazione mostra la formazione spontanea di stormi; i boids tendono ad avvicinarsi ad un centro di massa locale grazie al parametro “d” definito in precedenza, formando uno stormo all’interno del quale seguono traiettorie simili, coerentemente con le regole di separazione, allineamento e coesione. La variazione del colore dei boids è indice del modulo della velocità, maggiore è la velocità del boid maggiore è l’intensità del rosso. L’interazione col mouse ha evidenziato la corretta applicazione della forza attrattiva o repulsiva, con risposta immediata da parte dei boids. Quest’ultima, assieme alla gestione a runtime dell’aggiunta e della rimozione dei boids, permette di osservare come varia l’evoluzione del sistema al mutare delle condizioni.

Strategia di test

Per garantire l’affidabilità e la correttezza del comportamento simulato, abbiamo integrato una lista di test tramite Doctest. I test sono stati progettati per verificare le funzioni principali del progetto, valutandone l’output in diversi scenari, sia standard che casi limite.
Abbiamo adottato dei test con l'obiettivo di verificare la correttezza di:
- Funzioni matematiche di supporto (“diff_pos2”, “get_speed”, “add_inplace”, “is_neighbor”);
- Regole di movimento dei boids (“rule1”, “rule2”, “rule3”);
- Aggiornamento dinamico (“apply_neighbor_rules”, “apply_mouse_force”, “update_pos_vel”, “update”);
- Validazione dei limiti (“check_sides”, “limit_velocity”).

Uso dell’Intelligenza Artificiale

Abbiamo fatto ricorso a strumenti di Intelligenza Artificiale generativa (Chat GPT) per la gestione della sintassi riguardante la libreria SFML, sfruttata per la rappresentazione grafica. Sempre attraverso tale libreria l’AI ci ha fornito gli strumenti per poter implementare le funzioni che garantiscono l’interazione con il mouse. Infine è stata usata per la creazione di alcuni test aggiuntivi per verificare la correttezza del codice.