Boid Simulation

Notarpietro Sabrina Pivi Riccardo Schirripa Mattia

A.A. 2023/2024

1 Descrizione del progetto

Il progetto qui presentato riguarda lo sviluppo di una simulazione bidimensionale del volo di stormi, basata su un algoritmo sviluppato da Craig Reynolds nel 1986 [1]. Il codice in C++ e una copia di questa relazione sono disponibili sulla repository di Github citata in bibliografia [2].

Organizzazione del Codice

Il codice sorgente, sviluppato in C++, è stato suddiviso in cinque file: main.cpp, birds.cpp, twodimensional.cpp, test.cpp e trianglesfml.cpp; i file birds.cpp, twodimensional.cpp e trianglesfml.cpp presentano le definizioni delle classi e delle funzioni utilizzate. Nella repository sono presenti anche tre header file, contenenti le sole dichiarazioni delle classi e dei relativi metodi: essi presentano lo stesso nome dei relativi file sorgente, seguiti dal suffisso *hpp*. Infine sono stati inclusi file necessari: alla formattazione del codice (.clangformat), allo sviluppo di test (doctest.h), allo sviluppo collaborativo del codice (.gitignore) e alla compilazione di più file (CMakeLists.txt).

Struttura del Programma

Vettori Bidimensionali

La struttura *vec* e le dichiarazioni delle funzioni riguardanti l'ambito bidimensionale si trovano nel file twodimensional.hpp, mentre la loro definizione è presente nel corrispondente file twodimensional.cpp.
La struttura *vec* contiene due variabili di tipo double che corrispondono rispettivamente ai valori sull'asse delle ascisse (x) e sull'asse delle ordinate (y); per essa si sono implementati gli overloading dei principali operatori (+,-,*,/), in aggiunta ad altre due funzioni, una per calcolare la norma del vettore (*norm*) e l'altra per calcolare la distanza tra due vettori (*distance*).

La classe Boid

L'algoritmo di Reynolds ha per oggetto i *boids* (nome derivante da "bird-oid objects"), ovvero oggetti il cui singolo comportamento crea un fenomeno emergente: lo stormo. La classe *Boid* ed i suoi metodi sono definiti in birds.cpp; le sole dichiarazioni sono nel relativo header file. Ad ogni istanza della classe *Boid* sono associati due vettori bidimensionali: posizione e velocità. Sono presenti quattro funzioni relative a queste due grandezze: due, caratterizzate dal prefisso *get*, servono a rendere le variabili private accessibili all'esterno della classe, mentre le altre, con il prefisso *update*, aggiornano tali valori periodicamente. Vi sono poi una funzione per gestire il comportamento ai bordi (*borders*) e una per calcolare il centro di massa (*center_mass*). Per simulare correttamente il comportamento dello stormo, inoltre, ogni *boid* deve rispettare tre leggi: separazione, allineamento, e coesione.

Vicinanza, Separazione, Allineamento e Coesione

Per il corretto funzionamento delle tre leggi sopracitate, bisogna prima definire una funzione che permetta di capire se due *boids* sono vicini fra loro: tale compito è affidato al metodo *near*. Esso verifica se un altro *boid* è nel cerchio di raggio d centrato nel *boid* su cui viene chiamato.
In seguito si può procedere con *separation*, *cohesion* e *alignement*. Ogni legge restituisce una velocità che modifica quella del *boid* su cui viene chiamata: la separazione allontana le istanze troppo vicine, l'allineamento aggiusta gradualmente la direzione di movimento, affinché dopo un certo lasso di tempo sia la stessa per tutto lo stormo, e la coesione porta i *boids* ad avvicinarsi al centro di massa dei vicini. L'intensità delle leggi e il loro giusto bilanciamento dipende dai rispettivi parametri, richiesti in input dall'utente insieme al numero di istanze, ed assegnati tramite la funzione *set_parameters*.

La classe Flock

La classe *Flock* rappresenta il comportamento globale dei *boids*; anch'essa è definita nel file birds.cpp e dichiarata nel relativo header file. La classe *Flock* contiene le informazioni riguardanti i valori medi di posizione e velocità e i relativi errori sulla media di tali grandezze.

La funzione *createTriangle* ed SFML

I file trianglesfml.cpp e trianglesfml.hpp contengono rispettivamente la definizione e la dichiarazione della funzione *createTriangle*, usata per rendere gli uccellini di forma triangolare e fare in modo che la punta di

ognuno di essi indichi la direzione della velocità in cui si stanno muovendo. Nell'header file è inoltre inclusa la Simple and Fast Multimedia Library (in breve SFML [3]), necessaria per la generazione della visualizzazione grafica della simulazione.

Il file main.cpp e l'input

Il codice presente nel file main.cpp è responsabile dell'inizializzazione delle istanze dei *boids*, tramite la creazione del vettore *flock*, e di quella della classe *Flock*. Il numero di *boids* nel vettore, i parametri delle distanze di vicinanza e delle leggi sono richiesti in input all'utente sul terminale; in particolare essi rappresentano: la distanza alla quale agiscono l'allineamento e la coesione (d), che si consiglia impostare tra 70 e 80, la distanza alla quale agisce la separazione (d_s), che si consiglia impostare tra 15 e 20, il parametro che identifica l'intensità della separazione (s), che si consiglia impostare tra 0.04 e 0.06, quello per l'intensità dell'allineamento (a), che si consiglia impostare tra 0.001 e 0.002 e il parametro per l'intensità della coesione (c), che si consiglia impostare tra 0.002 e 0.003. Anche in questo file viene fatto uso della libreria SFML [3]: è qui che viene generata la finestra grafica 900 per 900 pixel, in cui l'angolo in alto a sinistra ha coordinata [0. ; 0.], mentre l'angolo in basso a destra [900. ; 900]. Sempre nel main si gestiscono le varie operazioni su di essa attuabili, quali la pulizia con uno sfondo azzurro per il cielo o il disegno dei triangoli che rappresentano i *boids*. Le istanze della classe *Boid* vengono inizializzate per via di numeri generati casualmente. Nel main.cpp è stata inclusa la libreria random, che permette di creare un oggetto random_device (nel codice chiamato r): questo oggetto preleva entropia da fonti hardware per generare un numero casuale non deterministico, successivamente usato come seme per la generazione di numeri pseudo-casuali uniformemente distribuiti. Sono quindi generati valori di tipo double fra 400. e 500. per le posizioni sia in x che in y, mentre fra -3. e +3. per le velocità sia in x che in y.

Simulazione Grafica e Output

Il progetto permette una visualizzazione grafica della simulazione: compilando ed eseguendo il codice, compare a schermo una finestra sulla quale si possono vedere i *boids* in movimento, rappresentati come triangolini neri su uno sfondo azzurro. Le punte dei triangolini sono orientate nell'effettivo verso di movimento del singolo oggetto. Infine vengono stampati a schermo (sul terminale) i valori restituiti dalla classe *Flock* nel seguente ordine orizzontale: posizione media in x, posizione media in y, errore sulla posizione media in x, errore sulla posizione media in y e poi la velocità media in x e in y e gli errori relativi in x e in y. Tali dati vengono incolonnati e preceduti dai relativi titoli.

2 Compilazione ed Esecuzione

Il programma è stato testato su Ubuntu 20.04 LTS e Ubuntu 22.04 LTS. Affinché la compilazione abbia successo è necessario avere installato CMake 3.16 o superiore e SFML 2.5 o superiore sulla propria macchina. La compilazione su più file è gestita dal sistema di build opensource Cmake [4]. Per creare una directory di build, il makefile CMakeLists.txt è già incluso all'interno del progetto; il comando da digitare è dunque, dopo essersi spostati nella directory dove è presente il progetto,

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
```

È sufficiente farlo solo una volta, purché non si modifichi il makefile. Per compilare il file principale è poi sufficiente digitare

```
cmake --build build
```

L'eseguibile per la visualizzazione grafica viene generato nella cartella build. Per eseguire il programma è quindi sufficiente spostarsi all'interno della stessa ed eseguire il comando

```
./boids
```

Sul terminale verranno richiesti i parametri citati nella sezione *Il file main e l'input*. Successivamente si aprirà una finestra con la rappresentazione grafica dei *boids* e sul terminale verranno stampati i valori medi di posizione e velocità con i relativi errori.
I test possono essere compilati ed eseguiti tramite comandi simili a quanto riportato sopra:

```
cmake --build build --target test
```

Poiché il file di test viene generato nella cartella debug, contenuta all'interno della cartella build, il secondo comando va digitato dopo essersi spostati al suo interno. Si è notato che, se il programma viene eseguito su una macchina virtuale come WSL, alla chiusura della finestra grafica potrebbe comparire il seguente errore:

```
==5124==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 44 byte(s) in 1 object(s) allocated from:
    #0 0x7ff1733b4c38 in __interceptor_realloc
    ↪  ../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:164
    #1 0x7ff16f099cbd
    ↪  (/lib/x86_64-linux-gnu/libGLX_indirect.so.0+0x39cbd)

SUMMARY: AddressSanitizer: 44 byte(s) leaked in 1
↪  allocation(s).
```

Esso sembra essere fornito dal LeakSanitizer, un avvertimento attivato nel makefile che notifica il memory leaking dovuto alla perdita di puntatori nell'allocazione dinamica della memoria. Inoltre, per la corretta visualizzazione grafica della simulazione tramite macchina virtuale potrebbe rendersi necessario l'utilizzo di un qualsiasi X server di uso gratuito (noi utilizziamo MobaXTerm).

3 Risultati

L'immagine riportata in [Figura 1] rappresenta le posizioni e le velocità medie nelle coordinate x e y dei *boids* durante una particolare simulazione. I grafici sono stati realizzati con ROOT [5], un pacchetto di analisi dati, sulla base di una simulazione con i seguenti parametri: *number of boids*=50, d=80, d_s=15, s=0.05, a=0.002, c=0.003. I *boids* sono stati generati uniformemente all'interno di una zona centrata rispetto alla finestra. La posizione media risulta congruente con ciò che ci si aspetta, ovvero: 450 in x e 450 in y (il centro della finestra). La velocità iniziale, anch'essa generata da una distribuzione uniforme, con media 0, provoca l'iniziale valore nullo per le velocità medie. Successivamente i *boids* si sono riuniti e hanno iniziato a spostarsi verso il fondo della finestra (aumentando così la loro posizione media in y) e in contemporanea si sono leggermente spostati verso destra (aumentando la posizione in x). È importate sottolineare che lo stormo che si era creato viaggiava con velocità maggiore e in maniera più uniforme in direzione y, per questo la velocità media lungo quell'asse risulta maggiore e con incertezza inferiore rispetto all'incertezza su x. I *boids* al trentesimo secondo hanno raggiunto il confine della finestra e hanno cambiato posizione, a causa dell'implementazione dello spazio toroidale; questo graduale cambiamento delle posizioni causa un crollo della posizione media in y e amplifica l'errore sulla posizione (lo si può denotare dal colore blu presente nei grafici).
Si consigliano i parametri usati per questa simulazione per provare il progetto.

4 Strategia di Testing

La correttezza del codice è stata verificata utilizzando il framework di Doctest. Tutti i test si trovano nel file test.cpp. In questo file si verifica la correttezza dei metodi e delle funzioni libere presenti nei file twodimensional.cpp e birds.cpp. I SUBCASE all'interno dei TEST_CASE sono disposti in ordine di complessità. Vengono richiamati in questa descrizione soltanto i test sui metodi più importanti: *near*, *separation*, *cohesion* e *alignment*, tutti appartenenti alla classe *Boid*. È stata posta particolare attenzione al metodo *near*, fondamentale per il corretto funzionamento di tutti i metodi più importanti. Sono stati testati tutti i casi limite di vicinanza tra *boids* (per esempio quando più *boids* sono sovrapposti oppure la vicinanza di un *boid* con se stesso), e dato che il metodo restituisce un valore booleano sono stati controllati sia i casi in cui il valore restituito è *true* che quelli in cui il valore è *false*. Per la separazione si sono effettuati diversi test con distanze diverse e diversi valori per il *boid* a cui vengono applicate le leggi ; inoltre in ogni test sono stati cambiati il numero e i valori degli altri *boid* appartenenti allo stormo. Lo stesso metodo è stato usato anche per le altre leggi.

Riferimenti bibliografici

[1] `https://www.red3d.com/cwr/boids/`.

[2] Repository di github: `https://github.com/rpivi/boids_simulation`.

[3] Sfml: Simple and fast multimedia library `https://www.sfml-dev.org/`.

[4] Cmake: `https://cmake.org/`.

[5] Root: `https://root.cern.ch/`.

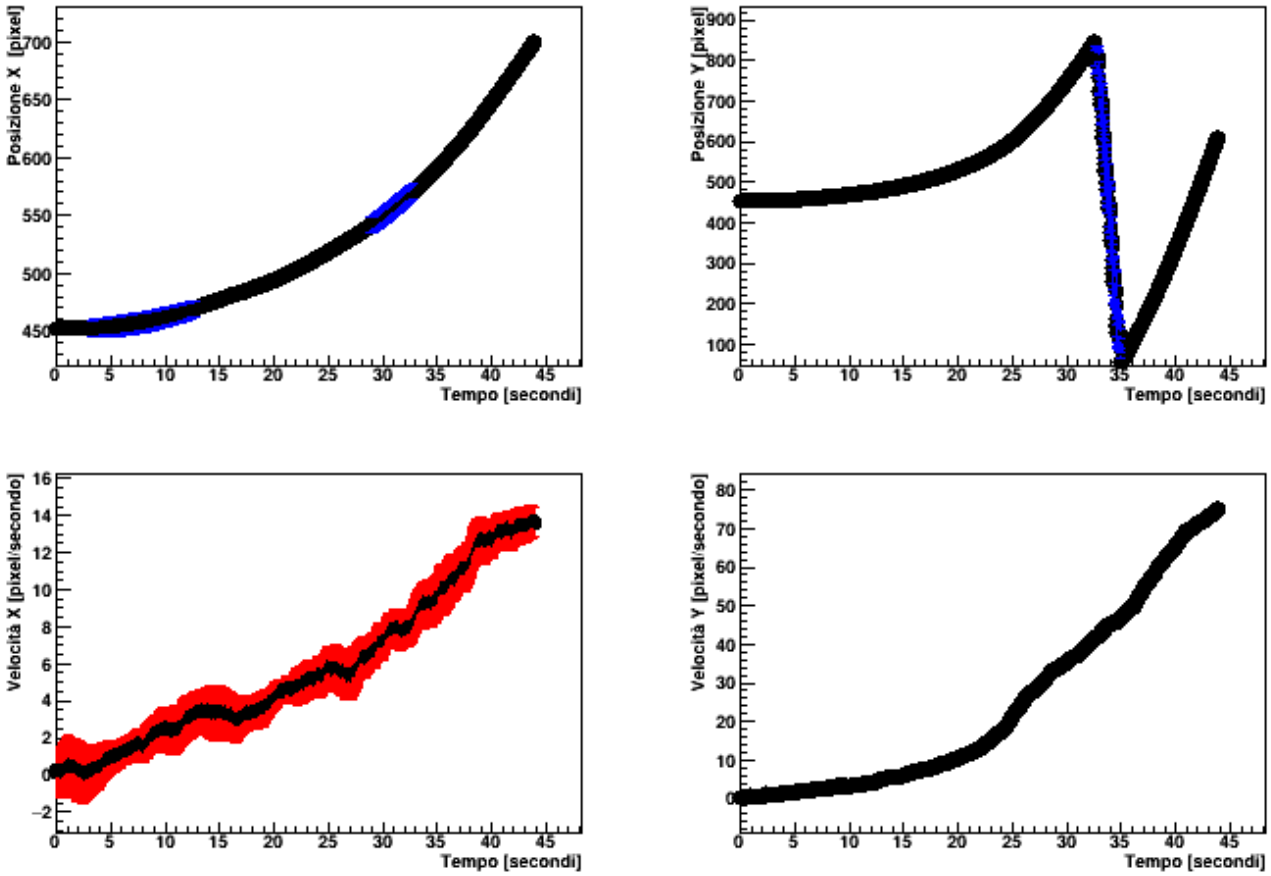

Figura 1: Grafici ottenuti con ROOT, dove è possibile osservare la posizione e velocità media in funzione del tempo per una particolare simulazione. Per questa prova sono stati inizializzati i parametri di input: *number of boids*=50, d=80, d_s=15, s=0.05, a=0.002, c=0.003. I valori delle medie sono rappresentati in nero, mentre l'incertezza è identificabile dal colore blu per le posizioni e dal colore rosso per le velocità.