FLOCK SIMULATOR

Francesco Cavina - Tito Barbieri

June 2024

Sommario

Abbiamo scelto la traccia proposta "Simulazione del comportamento di stormi" aggiungendo alcune features da noi ideate. Abbiamo fatto uso della libreria grafica SFML per mostrare a schermo l'output del nostro codice, del pacchetto sfml-widgets per la creazione di una GUI personalizzata. Per compilare e buildare il nostro codice abbiamo usato CMake, mentre per eseguire i test abbiamo utilizzato DOCTEST. Link alla repository di git che abbiamo utilizzato: GitHub Repository.

1 Cosa fa il programma

Il programma permette di simulare il comportamento di uno stormo di uccelli in volo in uno spazio bidimensionale. La traiettoria di ogni uccello è determinata da tre regole:

- separazione: l'uccello si allontana dai suoi vicini.
- allineamento: l'uccello tende ad allinearsi alle traiettorie dei suoi vicini.
- coesione: l'uccello si muove verso il baricentro dei suoi vicini.

Viene chiesto in input il numero di uccelli che si vuole spawnare *n_birds* e la dimensione dello spazio in cui avverrà la simulazione *box_size*. La nostra implementazione permette di modificare in tempo reale l'effetto più o meno marcato di ogni regola servendosi di un apposito menù. Nello specifico, i parametri che possono essere modificati durante la simulazione sono i seguenti:

- d distance: equivale al raggio dell'area circolare con centro nella posizione dell'uccello, che individua gli uccelli vicini che vanno considerati nel calcolo delle regole di allineamento e coesione.
- d_s distance: equivale al raggio dell'area circolare con centro nella posizione dell'uccello, che individua gli uccelli vicini che vanno considerati nel calcolo della regola di separazione.
- s factor: definisce l'effetto più o meno marcato della regola di separazione.
- a factor: definisce l'effetto più o meno marcato della regola di allineamento.
- c factor: definisce l'effetto più o meno marcato della regola di coesione.
- *max_bird_velocity*: definisce la velocità massima a cui può viaggiare un uccello.

Al fine di visualizzare meglio come stiano venendo modificati i parametri d e d_s, è possibile attivare un'apposita opzione che mostra a schermo la circonferenza di raggio uguale alle distanze in questione (Figura 1).
Il programma stampa a schermo la velocità media degli uccelli e distanza media uccello-uccello con relativi errori e salva in un file di testo una tabella contenente tutti i valori associati al tempo di simulazione trascorso (Figura 2).

2 Descrizione formato di input e output

Una volta eseguito il programma verrà mostrata a schermo una finestra (Figura 1) che richiede l'inserimento di due valori numerici interi: il numero di uccelli *n_birds* da spawnare e la dimensione della box *box_size* in cui avverrà la simulazione. I due valori possono essere impostati solo in un range fissato: 3-1000 per *n_birds* e 50-200 per *box_size*. Se l'input fornito non è corretto, viene stampato a schermo un errore e il programma rimane bloccato fino a quando l'input fornito non sarà corretto. Se l'input è corretto, la finestra si chiude, se ne apre una seconda (Figura 3) e ha inizio la simulazione. La finestra di simulazione presenta un menù interattivo con il quale è possibile modificare in tempo reale i parametri dello stormo; in questo caso non è necessario effettuare un controllo sull'input poichè l'impiego degli slider-widget (Figura 3, in alto a destra) permette di avere la certezza che i valori assegnati ai parametri siano dentro ad un range consentito.
All'esecuzione del programma viene inoltre creato, se non già esistente, un file di testo denominato *statistics_at_time_table.txt*. Il file di testo presenterà una tabella con tre colonne: la prima per la velocità media con relativo errore; la seconda per la distanza uccello-uccello media con relativo errore; e l'ultima per il tempo di simulazione trascorso (Figura 2). Durante l'esecuzione la tabella viene continuamente aggiornata: ad ogni aggiornamento del flock viene aggiunta una riga con i nuovi dati.

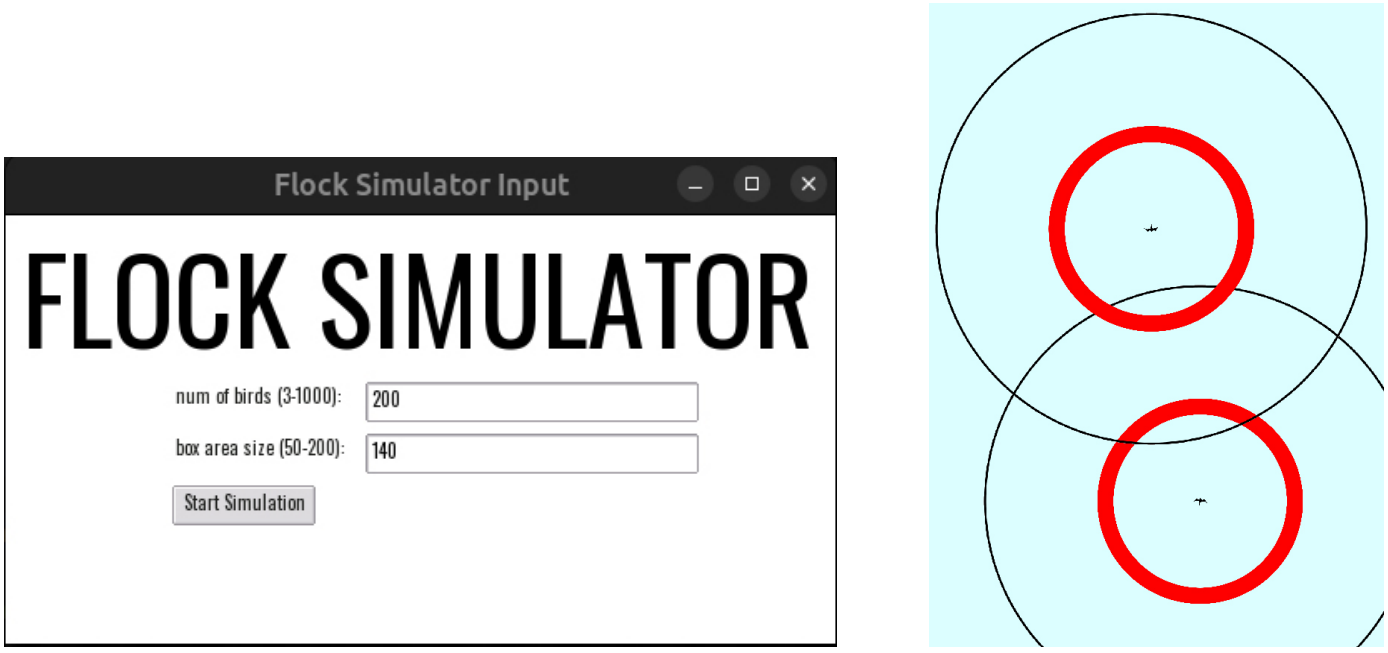

Figura 1: A sinistra: finestra di input. A destra: visualizzazione di d e d_s.

Average Velocity	Average distance	Time
71.17 +/- 0.24	80.26 +/- 1.81	0.001000
71.21 +/- 0.24	78.40 +/- 1.93	0.002000
71.25 +/- 0.24	79.26 +/- 1.86	0.003000
71.30 +/- 0.24	81.35 +/- 1.91	0.004000
71.34 +/- 0.24	83.06 +/- 2.10	0.005000
71.38 +/- 0.24	84.85 +/- 2.45	0.006000
71.42 +/- 0.24	85.91 +/- 2.60	0.007000
71.46 +/- 0.24	86.83 +/- 2.71	0.008000

Figura 2: Tabella creata nel file di tes to.

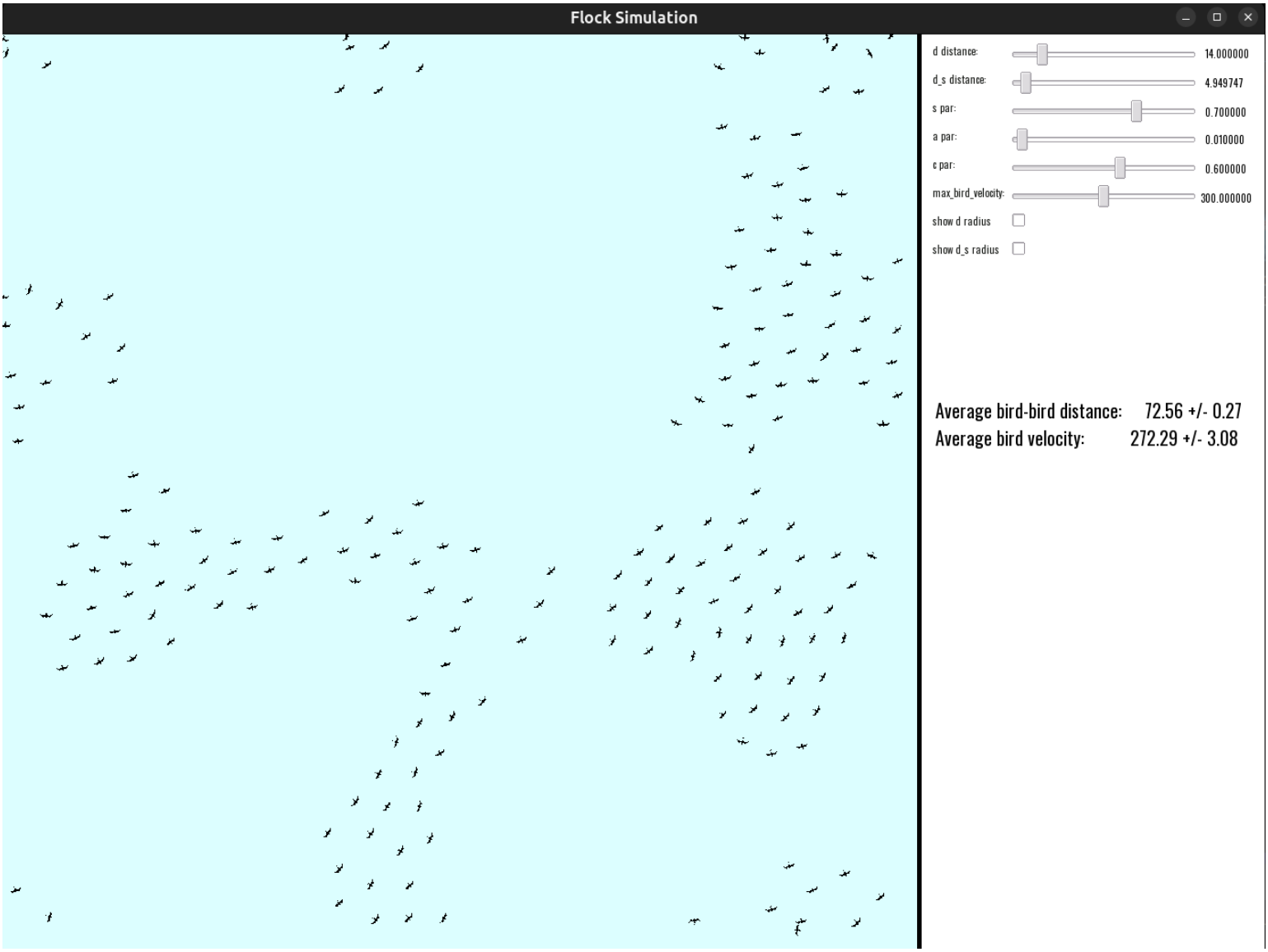

Figura 3: Finestra di simulazione.

3 Descrizione delle scelte progettuali ed implementative

Per gestire la simulazione nello spazio bidimensionale, abbiamo creato una struct *Vector2D* contenente due variabili di tipo double (per le coordinate x e y) e alcuni operatori utili per comparare vettori, sommare/sottrarre vettori o moltiplicare vettori per scalari. Fatto ciò, abbiamo creato una classe *Bird* che contiene le informazioni relative ad ogni uccello: un indice intero identificativo (unico per ogni uccello), la sua posizione e la sua velocità (entrambe variabili di tipo *Vector2D*).
Le varie funzioni che ci permettono di determinare il moto degli uccelli in accordo con le tre regole precedentemente menzionate (*separation_rule()*, *alignment_rule()* e *cohesion_rule()*) sono dichiarate in *flock.hpp*, dove è presente anche la classe *Flock*, contenente tutti i parametri dello stormo, un metodo per generare *n_birds* uccelli con posizioni e velocità randomiche e un metodo pubblico *update_birds_position()* che applica le regole dello stormo iterando su ogni uccello presente in *birds_*, vettore privato di *Flock* che raccoglie tutti gli uccelli che sono stati generati. I parametri di *Flock* devono avere un valore compreso in un certo range di ammissibilità, differente per ogni parametro e da noi scelto in

base alle prove che abbiamo fatto eseguendo il programma più volte.
Per gestire la GUI del nostro programma abbiamo creato le classi *FlockInputGUI* (per la finestra iniziale di input) e *FlockSimulationGUI* (per la finestra di simulazione). *FlockSimulationGUI* contiene tutto il necessario affinchè ciò che viene computato da *Flock* venga mostrato a schermo, tra cui, ad esempio, la texture dell'uccello.

4 Istruzioni su come compilare, testare ed eseguire

Il programma necessita della libreria SFML, che può essere installata su Ubuntu con il comando seguente:

```
$ sudo apt-get install libsfml-dev
```

La libreria sfml-widgets è inclusa nel progetto in /third_party/sfml-widgets e oppurtunamente linkata con CMake, perciò non è necessario installarla. Per buildare il programma in modalità Debug:

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

Per buildare il programma in modalità Release (più performante):

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

Per eseguire i test:

```
$ build/test.t
```

Per eseguire il programma:

```
$ build/flock_simulation
```

5 Interpretazione dei risultati ottenuti

L'esecuzione del programma crea, se non già esistente, un file *statistics_at_time_table.txt* all'interno del quale vengono salvati, all'interno di una tabella, i dati statistici dello stormo di uccelli: velocità media degli uccelli e distanza media tra gli uccelli dello stormo con relativi errori per ogni aggiornamento del flock.
In questo modo è possibile vedere come evolvono queste variabili dello stormo. Abbiamo notato che, con lo scorrere del tempo (a patto che i parametri del flock vegano lasciati invariati durante la simulazione), la velocità media tende a stabilizzarsi attorno ad una velocità fissa con piccole fluttuazioni attorno ad essa. Lo stesso andamento è possibile osservarlo con la distanza media degli uccelli, i quali tendono a stabilizzarsi in modo da avere distanza media fissa, con piccole fluttuazioni nel suo intorno.

6 Strategia di test

Al fine di eseguire controlli al runtime in alcuni punti critici del nostro codice abbiamo fatto uso della funzione *assert()* (contenuta nello standard library header *cassert*) che segnala comportamenti non previsti (il controllo è eseguito solo nel caso in cui il codice venga compilato con l'opzione *-DCMAKE_BUILD_TYPE=Debug* di CMake). Al fine di testare le funzioni del nostro codice e essere certi che esse funzionino anche in casi limite, abbiamo utilizzato DOCTEST e scritto tutti i test all'interno del file *test.cpp*.

7 Segnalazioni

La libreria sfml-widgets causa diversi warning al momento della compilazione se vengono impiegate le compile flags fissate dalla consegna del progetto. Abbiamo silenziato tali warning con il comando:

```
target_compile_options(sfml-widgets PRIVATE "-w")
```

presente a *riga 47* di *CMakeLists.txt*.
In *gui.hpp* sono presenti alcuni frammenti di codice necessario al funzionamento del menù di sfml-widgets che non sono stati scritti da noi, bensì dal creatore di sfml-widgets (i frammenti in questione sono segnalati nel codice con dei commenti).