

Programmazione per la fisica
Relazione progetto finale

Riccardo Vignoli
Susanna Bellentani
Andrea Buonsanti

28/05/2024

Indice

1	Introduzione	1
2	Implementazione del modello	1
2.1	Compilazione ed esecuzione del programma	2
2.2	Test	4
3	Conclusioni	4

1 Introduzione

Il progetto presentato consiste nell'implementazione di una simulazione del comportamento degli stormi di uccelli in volo in uno spazio bidimensionale in linguaggio C++. Il comportamento degli agenti interagenti, detti *boids*, è regolato da tre regole di volo: *separation*, *alignment* e *cohesion*. Queste permettono ai boids di aggregarsi in uno stormo, evitando però di collidere tra loro o con ostacoli esterni. Il programma mostra il movimento dei boids nel piano tramite l'utilizzo della libreria grafica SFML.

2 Implementazione del modello

L'implementazione consiste in 7 file:

- Tre file header: `vector.hpp`, `boid.hpp` e `b_simulation.hpp`
- Quattro file .cpp: `vector.cpp`, `boid.cpp`, `b_simulation.cpp` e `main.cpp`

Viene definita la classe "Vector", che implementa uno spazio vettoriale bidimensionale in cui si muovono i boids. Per fare ciò sono stati implementati in `vector.cpp` i metodi che realizzano sia le principali operazioni tra vettori che quelle tra vettori e scalari. Inoltre sono presenti le funzioni `magnitude()` e `control()` che rispettivamente calcolano e limitano il modulo di un vettore.

Nella classe "Boid" ogni agente è stato caratterizzato da una posizione e una velocità per poterne descrivere il movimento nel piano. La regola di volo **Separation** evita la collisione tra uccelli ed agisce solo tra boid a una distanza massima definita da `sep_param`, molto minore rispetto al parametro `near_condition`. Quest'ultimo determina invece il raggio di azione delle funzioni **Cohesion** e **Alignment**. Un giusto rapporto tra i parametri sopra permette la creazione di uno stormo senza scontri tra boids; se per esempio inizializzassimo `sep_param = near_condition`, i boids non riuscirebbero ad aggregarsi e ad allineare le loro velocità. Per mantenere i boids in uno spazio limitato abbiamo aggiunto il metodo **Potential**: quando un boid si avvicina al bordo della finestra gli viene applicata una velocità nella direzione opposta, inversamente proporzionale alla distanza dal bordo. Le tre regole di volo e il comportamento ai bordi vengono applicati agli uccelli da `Apply_rules`. E'fondamentale la presenza di `speed.control` all'interno del metodo: limitando il modulo delle velocità dei boids si evitano comportamenti anomali e si permette alla libreria grafica SFML di rappresentare sempre correttamente gli agenti interagenti. Per quanto riguarda gli ostacoli è stata implementata la funzione `Apply_rules_Obs` che contiene la sola regola **Separation**, in questo modo questi oggetti verranno evitati dallo stormo.

Viene poi dichiarata la classe "Simulation" in `b_simulation.hpp`, che costruisce l'ambiente grafico in cui si muoveranno i boids. Utilizzando SFML, il costruttore definisce le dimensioni della finestra, la posizione in cui compare e il limite di frame al secondo. Il metodo principale della classe è `Run_Simulation`, che si occupa di disegnare nella finestra gli uccelli e gli ostacoli, oltre ad aggiornarne le posizioni e le velocità. Questo è possibile grazie all'associazione del boid come oggetto fisico al triangolo che lo rappresenta nella finestra grafica. Sono inoltre presenti quattro funzioni che permettono di calcolare la distanza media tra i boids, la velocità media dello stormo e le rispettive deviazioni standard, stampate a schermo ogni 60 frame da `Flock_Means`. In `main.cpp` vengono gestiti gli input dello user.

2.1 Compilazione ed esecuzione del programma

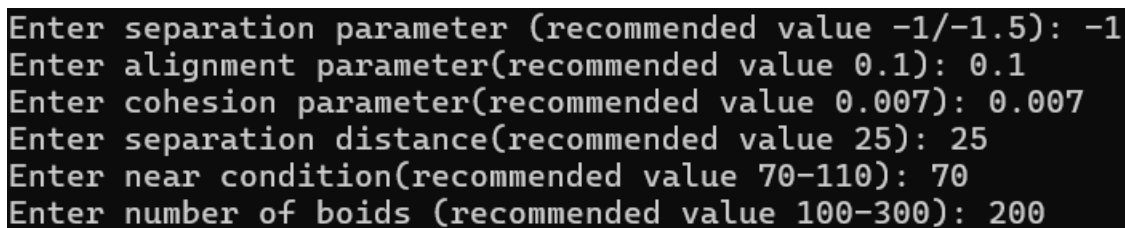
La compilazione viene effettuata con il compilatore g++ con i comandi necessari per SFML e le seguenti opzioni per rilevare problemi nel codice:

```
g++ -lsfml-graphics -lsfml-window -lsfml-system -Wall -Wextra -Wpedantic -Wconversion  
-Wsign-conversion -Wshadow -Wimplicit-fallthrough -Wextra-semi -Wold-style-cast  
-D_GLIBCXX_ASSERTIONS -fsanitize=address,undefined
```

Ad essere compilati saranno i file `vector.cpp`, `boid.cpp`, `b_simulation.cpp`, `main.cpp`:

```
g++ vector.cpp boid.cpp b_simulation.cpp main.cpp -lsfml-graphics -lsfml-window -lsfml-system  
-Wall -Wextra -Wpedantic -Wconversion -Wsign-conversion -Wshadow -Wimplicit-fallthrough  
-Wextra-semi -Wold-style-cast -D_GLIBCXX_ASSERTIONS -fsanitize=address,undefined
```

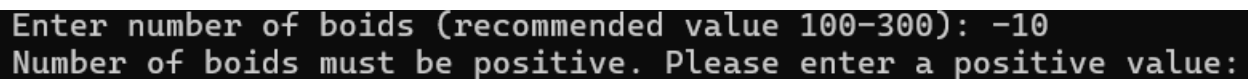
Se la compilazione va a buon fine, quindi senza alcun tipo di errore, warning o memory leak, si può procedere con l'esecuzione del programma con `a.out`. All'utente viene richiesto di inserire 6 parametri, attraverso questa schermata:



```
Enter separation parameter (recommended value -1/-1.5): -1  
Enter alignment parameter(recommended value 0.1): 0.1  
Enter cohesion parameter(recommended value 0.007): 0.007  
Enter separation distance(recommended value 25): 25  
Enter near condition(recommended value 70-110): 70  
Enter number of boids (recommended value 100-300): 200
```

Figura 2.1

Come visibile in Figura 2.1, viene data un' indicazione sul valore dei parametri, per ottenere il comportamento desiderato nella simulazione. Nel caso in cui l'utente dovesse inserire un valore non valido per un parametro, per esempio un numero di boids negativo, viene stampato a schermo un avviso come da Figura 2.2. Questa parte di codice viene gestita da `main.cpp`



```
Enter number of boids (recommended value 100-300): -10  
Number of boids must be positive. Please enter a positive value:
```

Figura 2.2

I boids vengono generati con posizione e velocità casuali nella finestra grafica, mostrando poi il comportamento desiderato con il passare del tempo. In Figura2.3 si mostra anche il comportamento dello stormo quando incontra un ostacolo. Ogni ostacolo viene generato cliccando con il tasto sinistro del mouse sulla finestra grafica. Se questo tasto viene mantenuto premuto sullo stesso punto dello schermo, si va a generare un grande numero di ostacoli che porta a comportamenti anomali da parte degli uccelli.

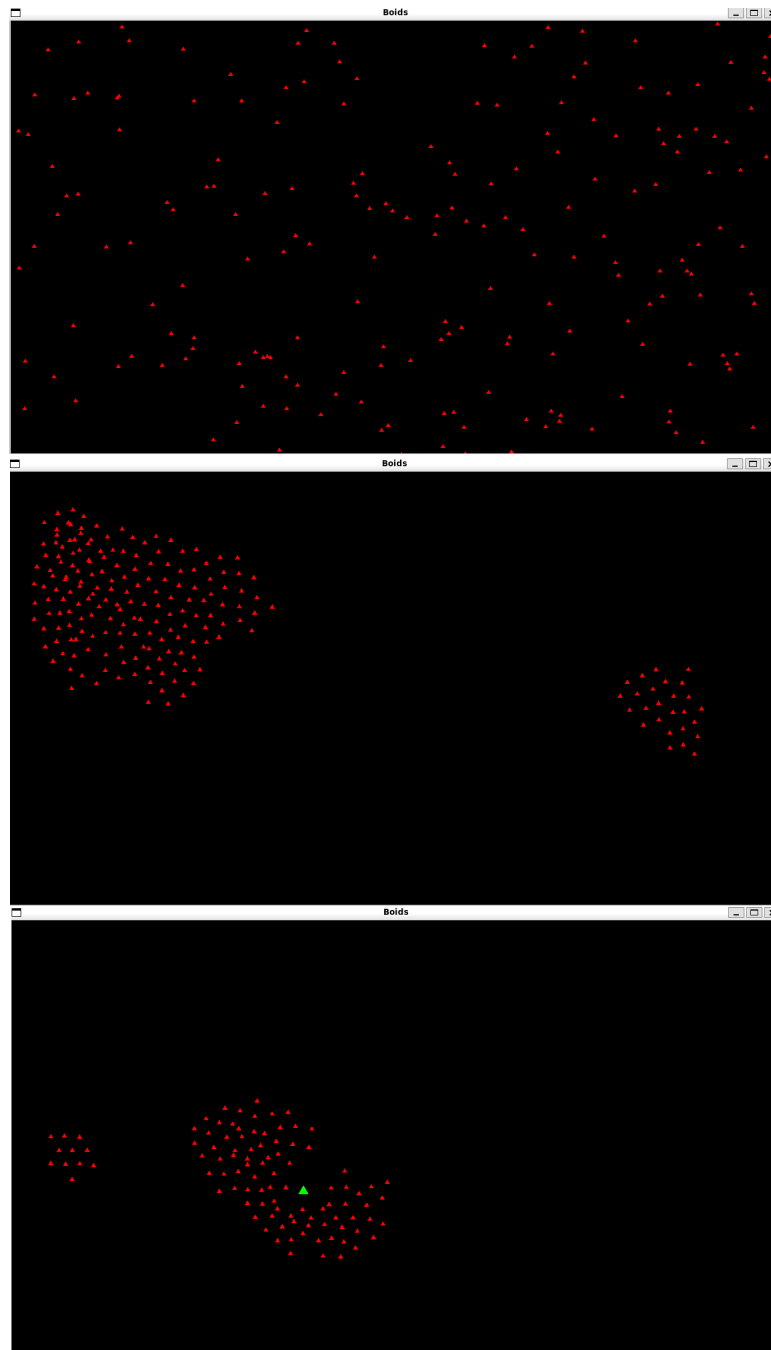


Figura 2.3: Dall'alto verso il basso: la posizione iniziale dei boids, la disposizione dopo circa 30 secondi, il comportamento dello stormo quando incontra un ostacolo

Oltre alla parte grafica vengono stampati a schermo la distanza media tra i boids, la loro velocità media e le rispettive deviazioni standard. Come visibile in Figura 2.4 i boids tenderanno ad aggregarsi in uno stormo, diminuendo la loro distanza media fino a stabilizzarsi attorno ad un valore costante.

```
Average Distance =460.834, Distance Std Dev = 347.789, Average Speed = 3.91127, Speed Std Dev = 0.147729
Average Distance =467.918, Distance Std Dev = 388.392, Average Speed = 3.71753, Speed Std Dev = 0.432711
Average Distance =468.013, Distance Std Dev = 404.356, Average Speed = 3.79328, Speed Std Dev = 0.363654
Average Distance =385.144, Distance Std Dev = 317.418, Average Speed = 3.82478, Speed Std Dev = 0.248707
Average Distance =218.201, Distance Std Dev = 151.116, Average Speed = 3.85147, Speed Std Dev = 0.223982
Average Distance =151.765, Distance Std Dev = 89.5701, Average Speed = 3.85525, Speed Std Dev = 0.236181
Average Distance =179.869, Distance Std Dev = 114.507, Average Speed = 3.81964, Speed Std Dev = 0.247585
Average Distance =154.794, Distance Std Dev = 91.988, Average Speed = 3.84181, Speed Std Dev = 0.287196
Average Distance =151.302, Distance Std Dev = 92.4618, Average Speed = 3.89387, Speed Std Dev = 0.135647
Average Distance =108.325, Distance Std Dev = 55.014, Average Speed = 3.84807, Speed Std Dev = 0.24885
```

Figura 2.4: Evoluzione dello stormo con parametri -1, 0.1, 0.009, 25, 110, 100

2.2 Test

Il progetto comprende il file di unit testing `test.cpp` in cui abbiamo testato alcune funzioni della classe `Vector` per verificarne il corretto funzionamento. Per quanto riguarda la classe `Boid` abbiamo verificato che le funzioni che riproducono le regole di volo "ritornino" un valore nullo se i boids a cui sono applicate si trovano ad una distanza maggiore di quella definita dai parametri `sep_param` e `near_condition`. Infine abbiamo testato le funzioni che calcolano medie e deviazioni standard.

3 Conclusioni

Il programma mostra in maniera efficace l'evoluzione di uno stormo a partire da una generazione casuale di boids. E' importante sottolineare come sia fondamentale una corretta scelta dei parametri per ottenere il risultato desiderato. Tra le criticità rilevate c'è la difficoltà nel trovare dei parametri compatibili per le funzioni `Separation` e `Cohesion` così che i boids non si tocchino, ma riescano comunque ad aggregarsi. In particolare si può notare come a volte i boids tendano ad appaiarsi per qualche secondo per poi tornare ad avere un comportamento coerente. E' stato necessario aggiungere un fattore correttivo di 0.2 nella funzione `control()`, definita in `vector.cpp`, in quanto non siamo riusciti a risolvere un bias sulla direzione dello spostamento dei boids: applicando le regole di volo senza la correzione, i boids, nel corso della simulazione, tendono ad avere velocità nulla lungo l'asse x (ovvero a muoversi solo in direzione verticale), nonostante nell'implementazione non dovrebbe esistere una direzione di moto privilegiata. Nonostante i vari tentativi di risolvere questo problema, non siamo riusciti a trovare altra soluzione che aggiungere un fattore empirico che non rendesse privilegiata nessuna delle direzioni. La funzione `control()` risulta quindi:

```
void Vector::control() {
float maxspeed = 4;
if (magnitude() > maxspeed) {
    x = (x / magnitude()) * maxspeed;
    y = (y / (magnitude() + 0.2f)) * maxspeed;
}
}
```