Boids

F. Bianchi, A. Polverari, L. Prestifilippo, L. Zavoli

Settembre 2022

Indicazioni sul progetto

La parte strutturale del codice, dove è implementata la gestione dei boidi e degli stormi e la loro evoluzione nello spazio, è uguale per i quattro studenti. A. Polverari, L. Prestifilippo e L. Zavoli hanno poi implementato la parte di grafica in maniera differente rispetto a F. Bianchi.

1 Introduzione

Il progetto consiste nell'implementazione di una simulazione del comportamento di uno stormo di uccelli in volo in uno spazio bidimensionale, basandosi su un software di intelligenza artificiale realizzato nel 1986. Il moto dei boidi nello spazio dipende principalmente da tre regole che, a seconda delle interazioni tra i boidi, ne regolano le traiettorie: la "separazione" allontana un boide da quelli vicini, evitandone la collisione; l'"allineamento" permette a un boide di allinearsi alle traiettorie dei vicini; la "coesione" fa sì che un boide tenda a muoversi verso il baricentro dei vicini, in modo da ottenere un gruppo di boidi unito come uno stormo. Grazie a queste tre regole di base i boidi si comportano come uno stormo di uccelli in volo. Inoltre, è stata implementata la possibilità di avere diversi stormi, che grazie alla regola di separazione si evitano tra di loro, oltre ad un angolo di visione che permette ai boidi di riconoscere i vicini solo entro quell'angolo. È infine possibile visualizzare il volo dei boidi grazie a delle librerie grafiche.

2 Implementazione

2.1 Vector

È stata implementata la classe Vector per rappresentare i vettori bidimensionali, utilizzati in particolare per definire posizione e velocità di ogni boide nello spazio. Inoltre, si è effettuato l'overload degli operatori per avere le operazioni tra vettori.

2.2 Options

La classe Options contiene i parametri per le tre regole dello stormo:

- double distance
- double separation_distance
- double separation
- double alignment
- double cohesion

2.3 Flock

La classe flock rappresenta lo stormo di boidi nello spazio, e presenta tre variabili private:

- std::vector<Boid> boids_ : contiene tutti i boidi dello stormo.
- Options boids_options_ : contiene i parametri per la simulazione degli stormi.
- double alpha_ : rappresenta l'angolo di visione di ogni boide appartenente allo stormo.

La classe presenta anche diversi metodi:

- Flock(std::vector<Boid> const& boids, Options const& boids_options, double alpha)
  Flock(std::vector<Boid> const& boids, Options const& boids_options)

  Questi due metodi sono i costruttori della classe. Il primo costruttore inizializza le tre variabili private, a partire dagli oggetti boids, boids_options e alpha. Nel caso in cui non si voglia utilizzare l'angolo di visione, è stato implementato anche il costruttore a due parametri, che permette di costruire un oggetto di tipo Flock inizializzando di default alpha_ a 180 gradi. In questo modo, la funzione view_neighbours e get_neighbours_of, implementate nel file Rules.hpp, si comportano allo stesso modo.

- std::vector<Boid> get_boids() const
  Options get_options() const
  double get_alpha() const

  Questi tre metodi restituiscono le variabili private di un oggetto di tipo Flock.

- int size() const
- void add(Boid const& boid)
- void evolve(double delta_t, double max_speed, double min_speed)

  È il metodo che fa evolvere lo stormo di un certo tempo pari a delta_t. Al suo interno vengono utilizzate le funzioni, definite nel file Rules.hpp, che applicano le tre regole di base: in questo modo vengono modificate la posizione e la velocità di ogni boide presente nello stormo, a seconda di quale siano i boidi vicini, ottenuti tramite la funzione view_neighbours. Inoltre, i parametri in ingresso di velocità massima e minima vengono utilizzati per chiamare la funzione speed_control, definita anch'essa nel file Rules.hpp, per controllare la velocità dei boidi. Una volta chiamato il metodo, tutti i boidi saranno quindi evoluti contemporaneamente.

- Vector get_distance_mean_RMS() const
  Vector get_speed_mean_RMS() const

  Il primo metodo considera le distanze tra tutti i boidi presenti in uno stormo, e restituisce un vettore bidimensionale che ha come componenti la distanza media tra i boidi e la sua deviazione standard; analogamente il secondo metodo restituisce un vettore con velocità media e sua deviazione standard.

2.4 Rules

Il file rules contiene la definizione delle regole di volo dei boidi:

- std::vector<Boid> get_neighbours_of(Flock const& flock, Boid const& boid)

  La funzione restituisce un std::vector contenente i vicini di boid appartenenti allo stormo flock. All'interno della funzione si calcola la distanza tra boid e ogni boide dello stormo, e se questa distanza è minore del parametro distance della classe Options viene considerato un suo vicino

- std::vector<Boid> view_neighbours(Flock const& flock, Boid const& boid)

  La funzione è definita a partire dalla precedente get_neighbours_of, che restituisce tutti i vicini di boid. View_neighbours restituisce invece tutti i vicini che rientrano nell'angolo di visione del

boide. Per far questo, la funzione calcola il prodotto scalare tra il vettore velocità di boid e il vettore distanza con ogni suo vicino, dividendo poi per il modulo, ottenendo il coseno dell'angolo compreso tra i due vettori. Infine, con la funzione arcocoseno calcola l'angolo compreso: se questo angolo è minore o uguale dell'angolo di visione, allora l'altro boide viene visto da boid.

- Vector separation(Options const& boid_options, Boid const& boid,
  std::vector<Boid> neighbours)

  Vector alignment(Options const& boid_options, Boid const& boid,
  std::vector<Boid> neighbours)

  Vector cohesion(Options const& boid_options, Boid const& boid,
  std::vector<Boid> neighbours)

  Sono le funzioni che applicano le tre regole di volo. Queste restituiscono tre oggetti di tipo Vector che, all'interno del metodo evolve di Flock, vengono sommati alla velocità iniziale di boid per ottenerne la nuova velocità.
  La funzione separation si occupa di applicare la regola di separazione. Si considerano tutti i vicini di boid che rientrano entro una distanza minore o uguale al parametro separation_distance della classe Options, e vengono sommati tutti i vettori distanza tra boid e quei vicini. Il vettore somma viene poi moltiplicato per l'opposto del parametro separation, che determina l'intensità della repulsione.
  La funzione alignment si occupa di applicare la regola di allineamento. Vengono sommate tutte le velocità dei vicini di boid, e poi divise per la dimensione del vettore neighbours. Successivamente si sottrae la velocità iniziale di boid, e si moltiplica per il parametro alignment, ottenendo così la correzione per l'allineamento.
  La funzione cohesion si occupa di applicare la regola di coesione. Si calcola il centro di massa dei boid vicini, come somma delle loro posizioni diviso la dimensione del vettore neighbours, al quale si sottrae la posizione iniziale di boid. Infine, si moltiplica per il parametro cohesion, ottenendo la correzione per la coesione dello stormo.

- void speed_control(Boid& boid, double max_speed, double min_speed)

  La funzione serve per controllare la velocità dei boidi, evitando che essi raggiungano delle velocità troppo elevate o troppo piccole. Vengono quindi fissate una velocità massima e una minima, da non superare: se questo accade, la velocità di boid viene divisa per il proprio modulo, per poi essere moltiplicata per la velocità massima o minima. In questo modo, chiamando la funzione dentro a evolve dopo le correzioni di volo, non viene modificata la traiettoria del boide.

2.5 MultiFlock

La classe MultiFlock rappresenta l'insieme di più stormi nello spazio. Questa presenta solo una variabile privata:

- std::vector<Flock> flocks_ : contiene tutti gli stormi presenti nella simulazione.

Presenta, inoltre, nove metodi:

- MultiFlock(std::vector<Flock> const& flock): è il costruttore della classe.
- std::vector<Flock> get_flocks() const

  std::vector<Boid> get_all_boids() const

  Il metodo get_flocks restituisce la variabile privata di un oggetto di tipo MultiFlock. Il metodo get_all_boids restituisce invece un vettore di Boid, nel caso in cui servisse applicare una stessa funzione a tutti i boidi della simulazione.
- int size() const

- void add(Flock const& flock)
- void evolve(double delta_t, double max_speed, double min_speed)

Il metodo evolve regola l'evoluzione di più stormi nello spazio. Viene preso come parametro il tempo di un'evoluzione, oltre alla velocità massima e minima che possono raggiungere i boidi. Per prima cosa, la funzione si occupa di mantenere separati gli stormi differenti: per ogni boide presente in uno stormo viene applicata la free function get_other_neighbours, che restituisce un vettore di Boid con tutti quei boidi che sono vicini ma appartenenti ad altri stormi; successivamente si applica la regola di separazione con questi "altri vicini". Infine, per ogni stormo viene chiamata la funzione evolve, definita nella classe Flock. In questo modo gli stormi vengono dapprima mantenuti separati, e solo successivamente evoluti secondo le regole di volo.

- void evolve(double delta_t, double max_speed, double min_speed,
  std::function<void(Flock&)> costraint_applier)

Questo secondo metodo evolve ha come parametro anche una funzione che prende come input un oggetto di tipo Flock. Lo scopo è permettere all'utente finale di poter modificare il comportamento in volo dei boidi, tramite l'aggiunta di una funzione arbitraria: ad esempio, l'aggiunta di una funzione che permetta agli stormi di rimanere all'interno dei bordi della finestra grafica. All'interno di questo metodo viene dunque chiamato l'evolve precente, e successivamente applicata la funzione costraint_applier.

- std::vector<Vector> get_all_distance_mean_RMS() const
  std::vector<Vector> get_all_speed_mean_RMS() const

Il primo metodo applica su ogni stormo, appartente all'oggetto di tipo MultiFlock, la funzione get_distance_mean_RMS definita nella classe Flock: con i vettori bidimensionali restituiti viene poi riempito un std::vector. In questo modo il metodo restituisce un vettore di Vector, dove ogni elemento ha come componenti la distanza media tra i boidi del rispettivo stormo e la sua deviazione standard. Analogamente, il secondo metodo applica la funzione get_speed_mean_RMS e restituisce un vettore di Vector, dove ogni elemento ha come componenti la velocità media del rispettivo stormo e la sua deviazione standard.

3. Strategia di test

L'obiettivo dei test è stato quello di verificare i punti critici del programma, oltre al buon funzionamento dei metodi delle varie classi e di alcune free functions.

3.1 Test della classe Vector

Per prima cosa è stata testata la classe Vector separatamente, in quanto i vettori bidimensionali sono necessari per istanziare un oggetto di tipo Boid. Sono stati testati tutti gli operatori, verificando il loro corretto funzionamento; in particolare, nel caso della divisione per zero, che non appartiene al dominio dell'operazione, si è verificato che venisse sollevata l'eccezione. Inoltre, si è anche testato il corretto funzionamento dei metodi x() e y(), oltre al costruttore con zero argomenti nella lista di parametri.

3.2 Test delle classi Boid, Flock, MultiFlock e delle regole di volo

Per quanto riguarda il tipo Boid, sono stati dapprima testati gli operatori per uguaglianza e disuguaglianza, e il costruttore a zero parametri. Inoltre, si sono testate tre funzioni definite nel file Rules.hpp: distance, applied_distance e speed. Queste vengono utilizzate per definire le funzioni delle regole di volo: dunque sono state testate verificando di ottenere ciò che ci aspettavamo dai

calcoli numerici. Sono state poi testate le funzioni get_neighbours_of e view_beighbours, applicate ad all'oggetto Boid b1. Per prima cosa, fissato il parametro distance della classe Options, si è visto che i boidi a una distanza maggiore di quel valore non venivano considerati vicini a b1, come ci si aspettava. Successivamente, tramite view_neighbours, si è verificato che solo alcuni dei vicini di b1 rientravano nell'angolo di visione pari a 90 gradi. Il test della funzione separation è stato realizzato verificando prima che la regola fosse applicata solamente a boidi a una distanza minore di separation_distance; successivamente si è visto che la correzione della velocità rispettava i valori ottenuti con un calcolo diretto, siano con 2 che con 3 boidi. Con la funzione alignment si è anche verificato che aggiungendo un vicino non visto, cioè fuori dall'angolo di visione, non contribuisse alla correzione della velocità.
Per la classe Flock sono stati testati il costruttore a due e tre parametri: in particolare, se il costruttore lavorasse bene inizializzando un oggetto di tipo Flock con un vettore di Boid vuoto, e che con quello a due parametri l'angolo di visione venisse inizializzato di default pari a 180 gradi. Inoltre, anche per il metodo evolve di Flock, si è verificato che i valori numerici restituiti dalla funzione rientrassero in quelli ottenuti tramite un foglio di calcolo.
Infine, anche per la classe MultiFlock è stato testato il metodo evolve, per l'evoluzione di più stormi nello spazio. È stato scelto un coefficiente di separazione molto elevato, in modo da rendere evidente la separazione tra due diversi stormi: si è fatto evolvere uno stormo prima da solo e poi con un altro stormo vicino, verificando che nei due casi si evolve in maniera differente.

4. Istruzioni per la compilazione

Sul terminale di comando, per compilare ed eseguire il programma si esegue il comando:

$ g++ vector.cpp boids.cpp main.cpp -std=c++17 -Wall -Wextra -fsanitize=address -lsfml-graphics -lsfml-window -lsfml-system && ./a.out

Per testare la classe Vector, si esegue il comando:

g++ vector.cpp vector.test.cpp -std=c++17 -Wall -Wextra -fsanitize=address && ./a.out

Per testare le classi Boid, Flock e MultiFlock e relativi metodi, e le funzioni contenute in rules.hpp, si esegue il comando:

g++ vector.cpp boids.cpp boids.test.cpp -std=c++17 -Wall -Wextra -fsanitize=address && ./a.out

Per la compilazione di graphics.cpp è necessaria la libreria grafica SFML, che si può liberamente scaricare al link: https://www.sfml-dev.org/.

5. Input e output del programma

Il programma si interfaccia con l'utente chiedendo di inserire in input i valori che definiscono il numero di stormi nella simulazione e il numero di boidi per ogni stormo, e i valori con cui inizializzare i parametri della classe Options della simulazione. Una volta inseriti questi parametri da terminale viene avviata la simulazione grafica; contemporaneamente viene riempito un file data.txt con informazioni riguardanti la simulazione in corso: ad ogni evoluzione, si stampano per ogni stormo media e deviazione standard delle distanze tra boidi, e media e deviazione standard delle velocità dei boidi.