Boids

Brugnami V., D'Agosta N., Furlanetto L.

Maggio 2022

1 Introduzione

Il progetto riguarda l'implementazione di una simulazione del movimento di stormi di uccelli e si basa sul software di intelligenza artificiale realizzato nel 1986 da Craig Reynolds. Nel caso più semplice possibile, i vari boidi si muovono liberamente nello spazio sottoposti solo a tre regole principali: la coesione, l'allineamento e la separazione. Queste tre regole di base fanno si che i boidi vadano a formare uno stormo che si muove in maniera simile a quelli reali, creando gruppi uniti e coesi ma facendo si che tra di loro vi sia sempre una distanza minima di separazione per evitare le collisioni. A questa versione di default, è stato deciso di aggiungere un predatore, in grado di "mangiare" i boidi, e regole per evitare il superamento di una velocità massima data e il rallentamento oltre una certa velocità minima. Inoltre grazie all'implementazione di confini é possibile scegliere tra tre spazi diversi: toroidale, cilindrico e rettangolare. Tra i tre, quello di default è quest'ultimo. Infine é stata aggiunta la possibilità di modificare i vari parametri, quali il numero di boidi, i fattori di separazione, allineamento, spazio ecc..., da terminale e la creazione di un file .csv nel quale vengono trascitte veocità medie e deviazioni standard delle velocità ad ogni frame (si faccia attenzione, il file viene riscritto ad ogni nuova simulazione).

2 Istruzioni

Per compilare senza l'ausilio di CMake si può eseguire nel terminale il comando:

> $ g++ -std=c++17 main.cpp vectors.cpp game.cpp boids.cpp -lsfml-graphics -lsfml-window -lsfml-system && ./a.out

Per compilare i test della classe dei vettori si può eseguire il commando:

> $ g++ -std=c++17 vectors.cpp ./tests/vectors.test.cpp -lsfml-graphics -lsfml-window -lsfml-system && ./a.out

Per compilare i test delle regole dello stormo si può eseguire il commando:

> $ g++ -std=c++17 vectors.cpp game.cpp boids.cpp ./tests/boids.test.cpp -lsfml-graphics -lsfml-window -lsfml-system && ./a.out

3 Strategia di Test

3.1 Test della classe dei vettori

Per prima cosa è stato necessario testare il funzionamento della classe "Vector2", essendo quella la base per il corretto comportamento della struct "Boid". Quest'ultima è infatti costituita da due variabili di tipo "Vector2". Si è iniziato pertanto testando tutte le operazioni tra vettori necessarie, e proprio perché più facile da testare gli overload dei vari operatori sono stati dichiarati come free functions all'interno di "vectors.hpp", e definiti poi in "vectors.cpp". Come test iniziale per ogni operatore sono stati scelti per comodità tre vettori, v1{1., 2.}, v2{2., 3.}, v3{4., 3.}, i quali sono stati combinati in modi diversi per ottenere più risultati, e nei casi in cui l'operazione avesse un certo dominio, come nel caso della divisione, con lo scopo di rompere il codice sono stati inseriti proprio quei valori esclusi dal campo di esistenza della funzione. Ciò è stato necessario per verificare che nel codice fossero presenti i vari runtime error del caso. Oltre alle varie operazioni, si sono poi testati i due metodi sempre dichiarati all'interno del file "vectors.hpp", ovvero la funzione "get_angle" e "magnitude", quest'ultima non presente come free function, ma dichiarata all'interno della parte pubblica della classe "Vector2". Nel corso di questi nuovi test si è dovuto utilizzare la classe Approx di doctest, in quanto sono presenti radici quadrate e coseni, e si è scelto di approssimare il risultato alla seconda cifra decimale, tramite la funzione "epsilon()". Anche qui si sono scelti dei vettori con il fine di rompere il codice, per essere sicuri di aver incluso all'interno del codice tutti i casi possibili.

3.2 Test delle regole dello stormo

Nel corso dell'evoluzione, i vari boidi sono sottoposti a determinate regole, le quali modificano velocità e posizione di essi. Per verificarne il corretto comportamento sono state quindi testate tutte le free functions rappresentanti queste regole, tutte dichiarate nel file "boids.hpp" e definite in "rules.hpp". Sono stati scelti quattro boids, che sono poi stati usati per testare tutte le varie funzioni nelle varie combinazioni. Per prima cosa è stata testata "get_neighbors", essenziale per capire a quali boidi applicare le regole. Dei quattro boid, solo b4 ci si aspettava rimanere escluso, in quanto ha una distanza dagli altri maggiore della distanza di visione di default. Infine nell'ultimo caso si è visto se l'angolo di visione funzionasse. Per fare ciò l'angolo è stato ridotto a 0.5 radianti, contro i 2.8 di default, ed è stato preso il boid b2, abbastanza vicino ma non rientrante nel campo di visione di b1. Si è quindi verificato che la dimensione del vettore ottenuto applicando get_neighbors fosse pari a 0. Sono stati poi testati i valori ottenuti dall'applicazione delle regole di coesione, allineamento e separazione. Poiché quest'ultima presentava una distanza di validità minore, in un caso sono stati considerati 3 boid, ovvero b1, b2 e b3, ma è stata scelta una distanza di 2, così da far si che nessun boid fosse abbastanza vicino da modificare la posizione di b1. Infine, si è testato il funzionamento delle free function "avoid_speeding" e "avoid_boundaries". Per il test della prima, la si è applicata su boid fermi e boid con velocità molto elevate, per verificare che queste fossero settate alle rispettive velocità minima e massima. Per la seconda funzione, invece, è stato necessario testarla su tutti e tre i diversi spazi. Per quanto riguarda lo spazio cilindrico, sono stati creati dei boidi con posizioni sulle x non comprese tra 0 e 1000. Applicando "avoid_boundaries", è stato confermato che la loro posizione veniva settata a 1000 laddove prima era negativa, e a 0 se prima la x del boid superava i 1000. Per quanto riguarda il test per le y, invece, sono stati presi dei boidi con posizioni vicine ai bordi superiore e inferiore, e si è testato come, applicando la funzione sei volte, le loro velocità li riportassero verso il centro dello spazio. Si è evitato di usare il metodo "evolve" per non modificare anche la posizione dei singoli boid. Lo spazio rettangolare invece punta a far andare verso il centro tutti i boidi vicini ai bordi. Quindi si è applicato lo stesso metodo di test dei bordi verticali per lo spazio cilindrico, ma lo si è esteso anche ai bordi laterali. Per quanto riguarda lo spazio toroidale, si è usato lo stesso metodo dello spazio cilindrico per i boidi aventi le x non comprese tra 0 e 1000, ma lo si è usato anche per quanto riguarda boid con le y non comprese tra le dimensioni dello spazio, verificandone il corretto settaggio.

4 Design

4.1 Classe Vector2

Questa classe è la rappresentazione di un vettore bidimensionale. Vi sono state implementate tutte le operazioni necessarie a questo progetto e in particolare il metodo get_angle() che, con una formula derivata dall'algebra lineare, permette di calcolare l'angolo (in radianti) compreso tra due vettori. (Testati tutti gli operatori in vector.test.cpp)

4.2 Classe Boids

La classe Boids contiene tutti i parametri dello stormo, che sono:

- separation (double)
- alignment (double)
- cohesion (double)
- distance (double)
- separation_distance (double)
- with_predator (boolean)
- view_angle (double)
- canvas_height (unsigned int)

- canvas_width (unsigned int)
- space (SpaceType)

Questa classe prevede 6 metodi:

- Boids(BoidsOptions const& boids_options) → È il costruttore della classe, prende come parametro un oggetto del tipo BoidsOptions, che viene passato come referenza costante, in modo da evitarne la copia. Nell'implementazione del metodo vi è la genereazione di numeri casuali che serve per inizializzare la posizione e la velocità iniziale di ogni Boid. È stato usato il metodo della standard library std::generate_n(), accoppiato con l'uso di un back_inserter per generare il vector di Boid. Infine vi è l'inizializzazione del predatore se ne è prevista la presenza nelle opzioni.
- void evolve() → Questo metodo viene chiamato all'interno del game loop (game.cpp , metodo Game::run()), facendo quindi evolvere lo stormo una volta a frame. All'interno di evolve() viene chiamato un altro metodo della classe Boids che è l'evolve_predator() la cui funzionalità verrà discussa nel prossimo punto. In evolve() viene fatto il loop su tutti i boidi e vengono individuati i vicini di ogni boid attraverso la funzione get_neighbors(). Successivamente ad ogni Boid vengono applicate le regole definite in "rules.hpp". Se è presente il predatore, viene tenuto un counter dei boid troppo vicini al predatore e il loro indice viene memorizzato in dead_boid_indexes. Dopo che il loop sui boidi termina vengono rimossi dal vettore i Boid "morti".
- void evolve_predator() → Viene fatto evolvere il predatore dopo averne identificati i vicini, al predatore non viene applicata la regole di separazione.
- void draw(sf::RenderWindow& window) const → È un metodo che disegna tutti i boidi e il predatore sulla finestra.
- Statistic calculate_statistics() const → Metodo che calcola media e varianza del campione, sia per la velocità sia per la posizione.

4.3 Rules

In questo file sono definite le regole alle quali sottostà lo stormo e sono:

- std::vector<Boid> get_neighbors(std::vector<Boid> const& boids, Boid const& boid, double distance, double view_angle) → Questa regola si occupa di individuare i boidi vicini ad un Boid dato. Viene fatto un loop su tutti i boidi e ne viene calcolata la distanza e l'angola tra la velocità del Boid del quale staimo cercando i vicini e il vettore distanza (tra la posizione del Boid e quella del possibile vicino), se entrambi i parametri sono soddisfatti il Boid viene inserito in un vettore che verrà poi ritornato dalla funzione. (Testata in boid.test.cpp:13-36).
- Vector2 apply_separation(std::vector<Boid> const& neighbors, Boid& boid, double separation_distance, double separation) → Questa regola si occupa di applicare la regola di separazione. Per prima cosa viene preso il vettore di vicini e viene ulteriormente ridotto mantenendo nel vettore solo i boidi che sono a distanze inferiori a separation_distance. Si è scelta questa strategia in modo da poter chiamare in evolve() solamente una volta get_neighbors() e utilizzarne il risultato per tutte le regole. (Testata in boid.test.cpp:38-46).
- Vector2 apply_alignment(std::vector<Boid> const& neighbors, Boid& boid, double alignment) → Questa regola si occupa di applicare la regola dell'allineamento come descritto nell'articolo, così che i boidi si muovano in maniera allineata. Tramite l'algoritmo std::accumulate si sommano tutte le velocità dei Boid appartenenti al vettore neighbors, e la si divide per la dimensione del vettore, trovando la velocità del centro di massa dello stormo. Si sottrae poi a questo valore la velocità del Boid dato e si moltiplica il risultato per una certa costante "alignment". (Testata in boid.test.cpp:48-56).
- Vector2 apply_cohesion(std::vector<Boid> const& neighbors, Boid& boid, double cohesion) → Questa regola fa in modo che i boidi tendano a raggrupparsi attorno al centro di massa dello stormo, così da essere coesi. Per prima cosa si individua quindi il centro di massa dello stormo dei boidi vicini, per questo alla free function viene passato direttamente il vettore "neighbors".

Si applica l'algoritmo std::accumulate, che somma tutte le posizioni dei vari boidi, e si divide il valore finale per la dimensione del vettore. Si sottrae poi al risultato la posizione del Boid dato, e si moltiplica il tutto per una certa costante "cohesion". Si ottiene così una velocità che punta verso il centro dello stormo. Inoltre, nel caso non ci fossero vicini, la velocità risultante è nulla. (Testata in boid.test.cpp:57-65).

- Vector2 avoid_predator(Boid& boid, int index, std::vector<int>& dead_boid_indexes, Boid const& predator, double separation_distance, double view_angle) → Questa regola si occupa di modificare la velocità di un certo Boid in modo da farlo allontanare dal predatore.

- void avoid_boundaries(Boid& boid, int const canvas_width, int const canvas_height, SpaceType space) → Questa regola serve per evitare che i boidi escano dallo schermo oltrepassando i bordi. Visto che sono stati implementati vari tipi di spazi, servono regole diverse a seconda del tipo di spazio scelto. Nel caso dello spazio cilindrico, quando il Boid supera i bordi laterali, viene "teletrasportato" nel lato opposto. Per quanto riguarda il bordo inferiore e quello superiore invece, viene settata al Boid una velocità così da farlo ritornare verso il centro. Lo spazio toroidale è stato strutturato in modo che, superato qualsiasi bordo, il Boid venga riposizionato nel bordo opposto, imponendo quindi la periodicità dell'area di simulazione. Infine, nello spazio rettangolare, se un Boid si avvicina troppo a qualunque bordo, gli viene settata una velocità tale da riportarlo verso il centro dello schermo. (Testata in boid.test.cpp:74-127).

- void avoid_speeding(Boid& boid, double max_speed = 5, double min_speed = 2) → Questa regola ha l'obiettivo di non far superare al Boid una certa velocità massima, nè far scendere la sua velocità sotto un certo valore minimo. Per fare ciò, si setta la velocità del boid alla velocità massima, tenendo però conto della direzione e del verso della velocità iniziale. Per far questo, la si divide per il suo modulo, in maniera tale da diventare un Vector2 formato da due versori. Moltiplicando poi questo per la velocità massima o minima, si ottiene il risultato desiderato. (Testata in boid.test.cpp:66-73).