Volo di stormi

Cardillo Emanuele
Dipietro Alessandro
Stefanini Lorenzo

16 June 2025

Scelte progettuali ed implementative

Il progetto si compone di quattro file principali:

- L'*header file* `boids.hpp`
- Il file di implementazione `boids.cpp`
- Il file contenente i test `boids.test.cpp`
- Il *main file* `main.cpp`

Il programma simula due categorie di entità:

- **Boid**: agenti autonomi che seguono regole di comportamento locali
- **Predatori**: agenti che inseguono i boid e li "catturano"

Per rappresentare posizione e velocità degli individui nel codice è stata definita una struttura `Vec2f`, composta da due variabili `float`. Sia boid sia predatori sono quindi modellati attraverso classi dedicate che utilizzano oggetti di tipo `Vec2f` come variabili membro per descrivere posizione, velocità e vettori di correzione del comportamento. Ogni boid è inizializzato con posizione e velocità generate casualmente, ed è inserito all'interno di un vettore. La scelta dell'impiego di quest'ultimo, invece di un array, permette di gestire la modifica della dimensione dello stesso, ad esempio tramite la funzione `erase()`.

Il comportamento degli uccelli è determinato da quattro regole:

- **Coesione**: mantiene lo stormo compatto;
- **Allineamento**: uniforma la direzione di movimento complessiva;
- **Separazione**: evita scontri e sovrapposizioni;
- **Fuga**: induce i boid ad allontanarsi dai predatori.

Per ciascun elemento, le correzioni vengono calcolate confrontando la sua posizione con quella di tutti gli altri. Questo approccio, seppur semplice da implementare, comporta un costo computazionale che cresce in modo quadratico con il numero di entità simulate, e quindi non è adatto per situazioni più complesse, in cui il numero di boid è particolarmente elevato.

A differenza di quanto accade per i boid, i predatori mantengono fissa la propria velocità, definita tramite la costante `VEL_PRED`. A variare sono soltanto le componenti direzionali, modificate affinché il predatore insegua il boid più vicino. Se la distanza tra loro è minore di un certo valore, definito da `CATCH_RADIUS`, il boid viene rimosso grazie alla funzione `erase()`. Questa logica, insieme a quella di separazione tra predatori, è implementata all'interno della funzione `evaluate_pred_correction()`.

La simulazione è visualizzata tramite la libreria SFML, in una finestra 600 $\times$ 600 pixel. Il comportamento ai bordi della finestra di entrambe le entità è regolato dalla funzione `limit_func(Vec2f &)` che implementa uno spazio toroidale: quando un'entità supera uno dei bordi della finestra, essa viene riposizionata sul lato opposto.

I boid vengono rappresentati come cerchi neri, mentre i predatori compaiono come cerchi rossi. Il framerate è fissato a 60 FPS per garantire una visualizzazione fluida. È stato conseguentemente definito un intervallo temporale `TIME_STEP = 1/60`, così che ogni aggiornamento della posizione sia coerente con il tempo simulato.

Istruzioni per eseguire il programma

Per eseguire il programma è necessario avere installato `CMake` e `Ninja`. Qualora tali componenti non fossero stati precedentemente installati, ecco i passaggi necessari:

Per MacOS

Aprire il terminale e installare tali componenti mediante `brew`, grazie al seguente comando:

```
$ brew install cmake ninja
```

A seguito dell'operazione è possibile verificare la corretta installazione delle due componenti tramite i seguenti comandi:

```
$ cmake --version
$ ninja --version
```

Per Linux

Aprire il terminale ed eseguire il seguente comando:

```
$ sudo apt install cmake ninja-build
```

Compilazione ed esecuzione

- Aprire il terminale e spostarsi nella cartella del progetto:

  ```
  cd /percorso/progetto2025
  ```

- Configurare l'ambiente di build con il comando:

  ```
  $ cmake -S . -B build -G "Ninja Multi-Config"
  ```

  Questo comando, grazie al file `CMakeLists.txt` presente all'interno della cartella, genera e configura una cartella di compilazione denominata `build`.

- Compilare il progetto:

  ```
  $ cmake --build build --config Debug
  ```

- Eseguire il programma principale:

  ```
  $ build/Debug/main
  ```

- Eseguire i test:

  ```
  $ build/Debug/boids.t
  ```

Descrizione input e output

Il programma richiede all'utente l'inserimento, tramite console, di alcuni parametri:

- **Parametri di comportamento** (valori `float` nell'intervallo [0,1]):
  - `s`, impiegato nel calcolo della velocità di separazione;
  - `a`, impiegato nel calcolo della velocità di allineamento;
  - `c`, impiegato nel calcolo della velocità di coesione.
- **Distanze di interazione** (valori `float` positivi):
  - `d`, distanza di interazione per le regole di allineamento e coesione;
  - `d_s`, distanza per la separazione, con `d_s < d`.

- **Numero di agenti:**
  - `N`, numero totale di boid nella simulazione (consigliato: valore moderato).

Se i parametri non rispettano i vincoli richiesti, il programma termina con un messaggio d'errore tramite eccezioni. Una combinazione consigliata dei parametri è:

```
s = 0.6, a = 0.3, c = 0.15
d = 130, d_s = 30
N = 100
```

Funzionalità interattive

Durante l'esecuzione, l'utente ha accesso a due funzionalità:

- **Visualizzazione delle statistiche**: premendo il tasto spazio, il programma stampa nel terminale la velocità media dei boid e la deviazione standard nelle due componenti.
- **Inserimento dei predatori**: è possibile inserire fino a 5 predatori facendo clic con il mouse nella finestra grafica. I predatori compaiono nel punto selezionato e interagiscono immediatamente con i boid vicini.

Implementazione dei test

Per validare il comportamento del sistema di simulazione dello stormo, sono stati implementati test automatici usando il framework `Doctest`.

I test verificano:

- Operazioni su `Vec2f`: operatori matematici e metodi `angle()`, `norm()`;
- Funzione `init_parametres()`: verifica il comportamento con valori fuori limite;
- Funzione `distance()`: tra boid e predatori;
- Metodi delle classi: `vel_max()` per `Boid`, `limit()` in entrambe le classi;
- Funzione `evaluate_boid_correction()`: testata sia con singole regole che con tutte attive;
- Comportamento dei predatori: inseguimento, separazione e fuga;
- Funzioni statistiche.

I test sono stati spesso valutati anche in casi limite (es. eccezioni), per rivelare eventuali errori.

Uso di sistemi di Intelligenza Artificiale

L'intelligenza artificiale è stata utilizzata per generare porzioni di codice, in particolare:

- La funzione `Vec2f &operator+= (Vec2f const &)`;
- La condizione dell'istruzione `if` alla riga 41/42 nel `main`:

  ```
  if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Space)
  ```

È stata poi consultata frequentemente per chiarimenti su metodi, algoritmi e per risolvere alcuni `warning` in fase di compilazione.