

Relazione Progetto Boids

Francesco Bartoli

Contents

1	Introduzione	2
1.1	Scopo	2
1.2	Installazione	2
1.2.1	Prerequisites	2
1.2.2	SFML and TGUI Installation	2
1.2.3	Clone the Repository	2
1.2.4	Build the Project	2
1.2.5	Running the program	2
2	Struttura del programma	3
2.1	Regole di volo	3
2.1.1	Separazione	3
2.1.2	Allineamento	3
2.1.3	Coesione	3
2.1.4	Repulsione	3
2.1.5	Interazione <i>On Click</i>	3
2.2	Files	4
2.2.1	constants.hpp	4
2.2.2	structs.hpp	4
2.2.3	boid.hpp/cpp	4
2.2.4	flock.hpp/cpp	4
2.2.5	random.hpp/cpp	4
2.2.6	statistics.hpp/cpp	4
2.2.7	graphics.hpp/cpp	5
2.2.8	switchbutton.hpp/cpp	5
2.2.9	gui.hpp/cpp	5
3	Interfaccia della simulazione	6
3.1	Option 1: Graphics and statistics	7
3.2	Option 2: Create Flocks	7
3.3	Option 3: Simulation Parameters	7
3.4	Key Controls	7
4	Testing	8
4.1	Eseguire i test	8
5	Descrizione dei risultati	8
6	Links	8

1 Introduzione

1.1 Scopo

Il programma ha come obiettivo quello di simulare in uno spazio bidimensionale il comportamento di stormi di uccelli in volo, che verranno indicati con il nome di *boids*.

1.2 Installazione

Le istruzioni su come compilare, testare, eseguire sono presentate nel README del progetto, riportato qui sotto:

Build instructions are for **Ubuntu 22.04**.

1.2.1 Prerequisites

SFML (2.5): Library for graphic representation.

TGUI (1.0): Library for graphic interface.

1.2.2 SFML and TGUI Installation

Install SFML:

```
sudo apt install libsFML-dev
```

Install TGUI:

```
sudo add-apt-repository ppa:texus/tgui
sudo apt update
sudo apt install libtgui-1.0-dev
```

1.2.3 Clone the Repository

```
git clone https://github.com/Evyal/boids.git
```

1.2.4 Build the Project

- Create the build directory

```
mkdir build && cd build
```

- Configure CMake in Release mode

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

- Build the project

```
cmake --build . -j$(nproc)
```

1.2.5 Running the program

```
./boids
```

2 Struttura del programma

I file del progetto sono organizzati in sottocartelle, nello specifico i `.cpp` si trovano nella `/source`, mentre i rispettivi header sono situati nella `/include`. La directory `/testing` contiene i file di testing, e infine in `/assets` sono presenti alcuni file necessari per il corretto funzionamento del programma.

2.1 Regole di volo

I *boids* si seguono delle regole di volo, che ne determinano il comportamento. Ad ogni istante, il programma modifica le velocità e le posizioni dei *boids* attraverso le seguenti formule:

$$\vec{v}_{bi} = \vec{v}_{bi} + \vec{v}_S + \vec{v}_A + \vec{v}_C + \vec{v}_R$$

$$\vec{x}_{bi} = \vec{x}_{bi} + \vec{v}_{bi} \cdot k$$

Dove k è un fattore di scala, mentre \vec{v}_S , \vec{v}_A , \vec{v}_C , e \vec{v}_R sono rispettivamente:

2.1.1 Separazione

$$\vec{v}_S = -s \sum_{j \neq i} (\vec{x}_{b_j} - \vec{x}_{b_i}) \quad \text{se} \quad |\vec{x}_{b_i} - \vec{x}_{b_j}| < d_s$$

2.1.2 Allineamento

$$\vec{v}_A = a \left(\frac{1}{n-1} \sum_{j \neq i} \vec{v}_{b_j} - \vec{v}_{b_i} \right) \quad \text{se} \quad |\vec{x}_{b_i} - \vec{x}_{b_j}| < i$$

2.1.3 Coesione

$$\vec{x}_c = \frac{1}{n-1} \sum_{j \neq i} \vec{x}_{b_j} \quad \text{se} \quad |\vec{x}_{b_i} - \vec{x}_{b_j}| < i$$

$$\vec{v}_C = c(\vec{x}_c - \vec{x}_{b_i})$$

Dove $n-1$ assume valori dipendenti dal numero di boid nel range di interazione, e non è un valore fisso dipendente dal numero di boids nello stormo.

E s, d_s, a, c, i sono parametri della simulazione, e nel progetto sono indicati con i nomi di *separationStrength*, *separationRange*, *alignmentStrength*, *cohesionStrength* e *interactionRange*.

2.1.4 Repulsione

Questa regola in termini di formula è analoga a quella della separazione e determina l'allontanamento tra *boids* di stormi differenti introducendo due nuovi parametri r, dr , che nel progetto sono indicati con i nomi di *repelStrength* e *repelRange*.

2.1.5 Interazione On Click

$$\vec{v} = \pm p \sum_j (\vec{x}_{b_j} - \vec{x}) \quad \text{se} \quad |\vec{x}_{b_j} - \vec{x}| < i$$

Anche questa regola ha una formula analoga a quella della separazione e permette all'utente di interagire con i boids. Il \pm è dovuto al fatto che questa interazione può essere sia attrattiva che repulsiva mentre \vec{x} è il punto in cui l'utente ha cliccato. Il parametro che determina l'intensità di questa interazione prende il nome di *clickStrength* all'interno del progetto.

2.2 Files

Tutti i file del progetto sono stati inseriti nel namespace `ev::`.

2.2.1 constants.hpp

Header file che definisce i valori delle costanti usate nel progetto raggruppandole nel namespace `constants`. Alcuni esempi sono limiti di posizione o velocità per i *boids*, parametri di interazione di default, o ulteriori valori fissi per l'inizializzazione degli elementi dell'interfaccia grafica. Uno dei principali vantaggi di questo approccio è la possibilità di cambiare valori riutilizzati in diverse unità del programma, tutti da un unico punto, conferendo una maggiore facilità della loro modifica.

2.2.2 structs.hpp

Header contenente la definizione di alcune `struct` designate prevalentemente ad impacchettare dei valori usati per inizializzare bottoni o altri elementi di interfaccia grafica. Trovano particolare utilità come parametri delle funzioni che inizializzano tali elementi, aiutando a mantenere il codice più pulito. Un esempio è la `struct TguiPar` che contiene i parametri per inizializzare un generico elemento di interfaccia grafica, da cui seguono alcune `struct` derivate, come `SlidersPar`. È presente un'unica `struct` che non ha a che fare con la parte grafica, e contiene le variabili che parametrizzano l'interazione dei *boids* negli stormi.

2.2.3 boid.hpp/cpp

File che si occupa dell'implementazione della classe `Boid` e di tutte le funzioni ausiliari necessarie per gestirne il comportamento. Oltre alle variabili private che contengono posizione e velocità dei *boids*, sotto forma di `sf::Vector2f`, sono presenti funzioni membro come *getters* e *setters* per accedere a quest'ultime. Vi sono anche funzioni esterne alla classe, che lavorano su elementi della classe, come per esempio `distance` che calcola la distanza fra *boids* o `checkMinimumSpeed` che ha il compito di assicurarsi che il boid in questione abbia una velocità compresa all'interno dei limiti prestabiliti, ed eventualmente modificarla. È importante sottolineare che la velocità limite non è stata intesa all'interno del progetto come un'invariante di classe, ma semplicemente un limite imposto durante la simulazione per assicurarsi un comportamento ordinato, infatti non è vietato che un *boid* possieda una velocità superiore al limite prestabilito, in quanto questo non invalida nessuna delle funzioni che lo utilizzano.

2.2.4 flock.hpp/cpp

File di implementazione della classe `Flock` che determina la struttura collettiva dei *boids* all'interno di uno stormo. La classe presenta numerose variabili statiche, condivise da tutti gli stormi, che costituiscono i parametri dell'interazione tra *boids*, e due variabili membro private, la prima un vettore di *boids*, in cui sono contenuti i boid che compongono lo stormo, e l'altra che determina il colore dello stormo. Il vettore di *boids* deve contenere almeno due elementi, e se si cerca di inizializzare un'istanza appartenente alla classe `Flock` con meno di due elementi, questo risulterà in un errore a *compile-time*. Per assicurarsi che la classe non permetta l'utilizzo dei suoi metodi su stormi con meno di due *boids*, sono stati inseriti multipli `assert()` all'interno del corpo di quest'ultimi in modo da evitare che si verifichino comportamenti indesiderati. La parte restante della classe è designata all'implementazione delle regole che determinano il comportamento degli stormi, e della funzione `updateFlock()` che si occupa di aggiornare lo stato dello stormo. Quest'ultima funzione ne ha una *overloaded*, in grado di prendere come argomento le velocità da sommare dovute alla repulsione con altri stormi.

2.2.5 random.hpp/cpp

File che si occupa della generazione di numeri casuali. Sono presenti una funzioni base per generare numeri interi casuali di tipo `int` o `float`, e funzioni più complesse per generare posizioni e velocità dei *boids* in range prestabiliti.

2.2.6 statistics.hpp/cpp

File che si occupa del calcolo delle statistiche restituite a schermo, riguardanti valori medi e deviazioni standard delle posizioni e velocità dei *boids*. Ad esempio, la funzione `calculateDistances()` calcola le distanze tra tutte

le coppie degli elementi di un vettore, mentre `calculateToroidalDistances()` fa lo stesso ma in geometria toroidale. Infine è presente una funzione per il calcolo della media e una per il calcolo delle deviazioni standard.

2.2.7 `graphics.hpp/cpp`

Breve file che contiene due funzioni. La prima si occupa di rappresentare graficamente i *boids*, posizionandoli ed orientandoli correttamente. La seconda costruisce un rettangolo (`sf::Rectangle`) prendendo come parametro una delle struct definite nel file sopracitato, con lo scopo di migliorare la leggibilità del codice nella creazione dell'interfaccia.

2.2.8 `switchbutton.hpp/cpp`

La classe `SwitchButton` introduce la funzionalità di un bottone che può trovarsi in due stati, non fornita da TGUI, e si è cercato di seguire le convenzioni di TGUI nella creazione delle funzioni membro. In particolare è presente un costruttore, che crea un oggetto della classe, ed una funzione `create()` che restituisce uno `std::shared_ptr` che punta all'oggetto.

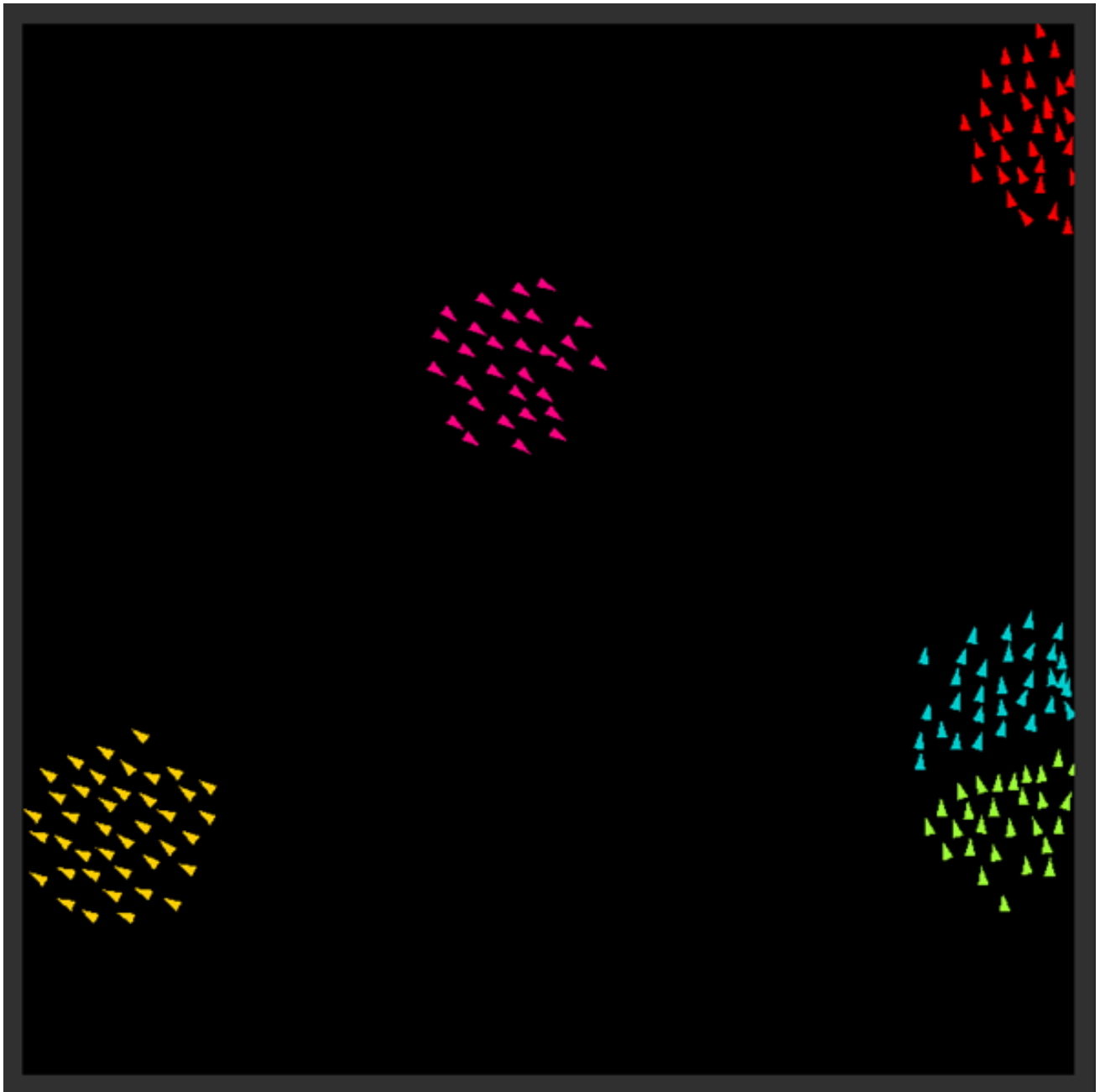
2.2.9 `gui.hpp/cpp`

In quest'ultimo file è definita la classe che contiene gli elementi necessari a realizzare l'interfaccia grafica del programma, e si occupa di coordinare gli altri file all'interno di essa. La classe contiene diversi membri privati, come ad esempio la `sf::RenderWindow`, per la raffigurazione grafica, l'elemento `tgui::Gui` per il funzionamento dei pulsanti, un vettore che contiene gli stormi attivi, e così via. Le funzioni al suo interno sono prevalentemente di *setup*, e sono organizzate in tre macro-aree, ciascuna con lo scopo di configurare la propria opzione di menù. Infine sono presenti ulteriori funzioni, come quella di creazione del menù (`createThreeWaySwitch()`), e quelle di gestione, creazione e distruzione degli stormi.

In ultima analisi, la classe `Gui` è responsabile del generale funzionamento del programma. Nel main è sufficiente costruire un elemento della classe, utilizzare il metodo `setup()` che lo configura correttamente e infine chiamare il metodo `run()` che contiene le istruzioni da seguire durante l'esecuzione.

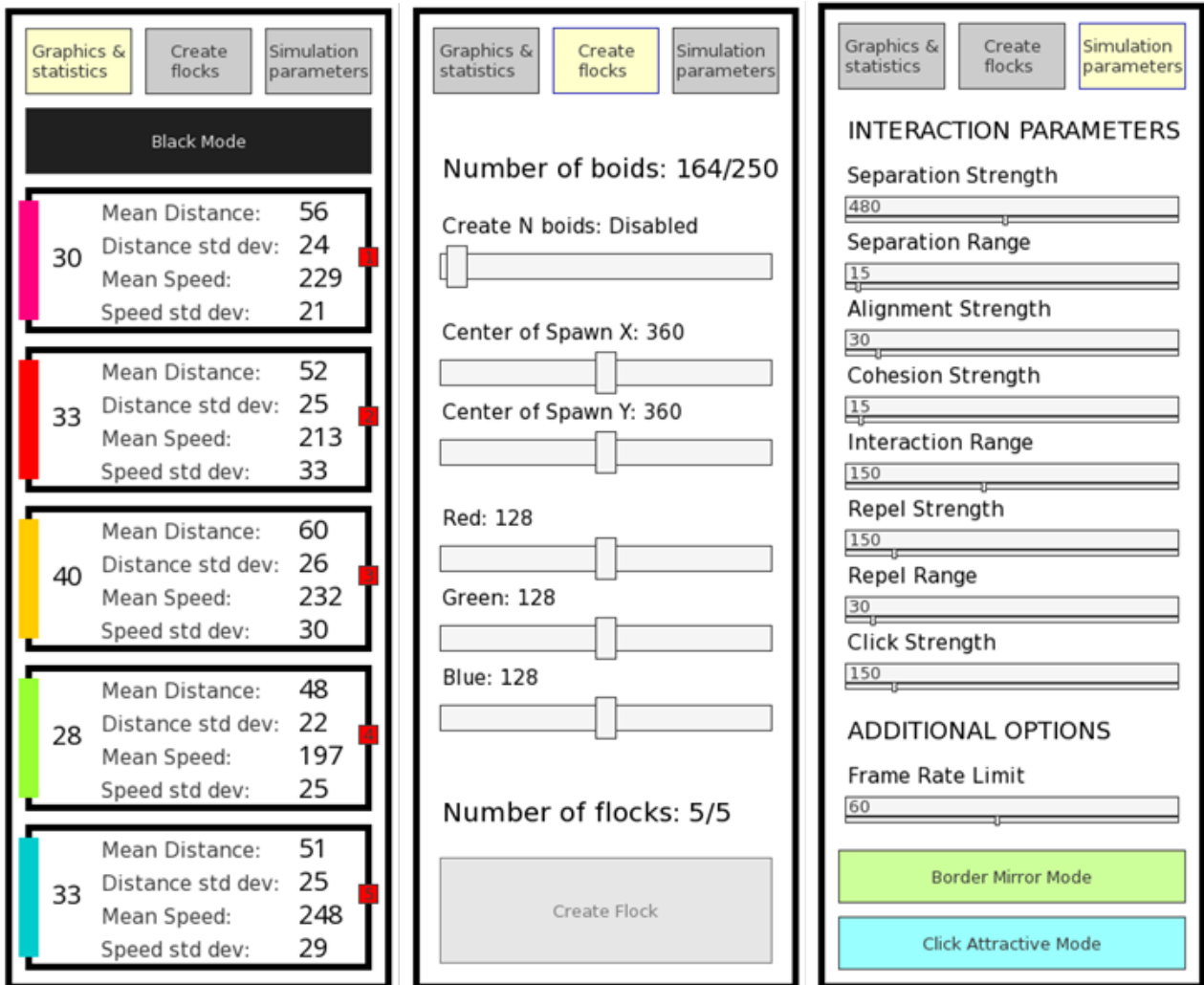
3 Interfaccia della simulazione

Anche questa parte derivante in parte dal README del progetto.



Features

- Real-time statistics display for each flock
- Interactive controls for adding or removing flocks
- Adjustable parameters for interactions between boids



3.1 Option 1: Graphics and statistics

- **Background color button:** Changes the colour of the background. (black and white)
- **Red numbered buttons:** Delete the corresponding flock.

3.2 Option 2: Create Flocks

- **Number of boids slider:** Selects the number of boids for a new flock.
- **Center of spawn sliders:** Select the spawn location of a new flock.
- **RGB sliders:** Select the color of a new flock. (Creating a white or black flock is disallowed because it would be invisible)
- **Create flock button:** Creates a new flock if there is enough space. (Max 250 boids; Max 5 flocks)

3.3 Option 3: Simulation Parameters

- **Interaction parameters sliders:** Change the values of the parameters of the rules that determine the movement of *boids*.
- **Border mode button:** Changes the behaviour of *boids* at the borders. (mirror or toroidal)
- **Click mode button:** Changes the interaction on click. (attractive or repulsive)

3.4 Key Controls

- **Left Click:** Interact with boids, attracting or repelling them to cursor.
- **Space Bar:** Pause/Resume simulation.

4 Testing

Tutti i file incaricati dell'implementazione di parte della logica del programma hanno un corrispettivo file di testing. Più precisamente, sono presenti i seguenti: `testboid.cpp`, `testflock.cpp`, `testrandom.cpp` e `teststatistics.cpp`.

Attraverso i test si è cercato di controllare che le classi, i metodi delle classi e le funzioni introdotte fossero esenti da errori e mostrassero il comportamento atteso. Sono stati eseguiti test in casi semplici per poter stabilire il funzionamento corretto del codice, e anche in alcuni casi particolari quando ritenuto necessario.

Il framework che si è utilizzato per creare le testing unit è `doctest.h`, il cui file è incluso nella cartella nel progetto (`/assets/doctest.h`). Questa libreria è in grado di generare autonomamente un main e permette l'esecuzione dei test semplicemente includendo il file sopracitato.

Un altro strumento utilizzato per controllare lo svolgimento corretto del programma sono degli `assert()`, prontamente inseriti all'interno di alcune funzioni per assicurarsi che determinate condizioni fossero verificate.

4.1 Eseguire i test

Per potere eseguire i test è necessario trovarsi nella cartella dove vengono prodotti gli eseguibili dei file precedentemente menzionati. Immaginando di trovarsi nella cartella principale dov'è contenuto il progetto, sarà sufficiente eseguire i seguenti comandi da terminale:

```
cd build/testing
```

E digitare il comando corrispondente al test che si vuole eseguire:

```
./testboid
./testflock
./testrandom
./teststatistics
```

O eventualmente è possibile eseguirli tutti in una volta utilizzando il seguente comando

```
ctest
```

5 Descrizione dei risultati

Nel caso della modalità toroidale ai bordi, i *boids* si raggruppano nei rispettivi stormi e tendono ad assumere valori di distanza media proporzionali al numero di elementi che lo compongono, e lo stesso vale per la deviazione standard delle distanze. Anche per quanto riguarda le velocità, esse raggiungono valori stabili indipendentemente dalle dimensioni dello stormo.

Diversamente accade nel caso della modalità a specchio (*mirror mode*). Infatti, i *boids* una volta ai bordi sono soggetti a un boost conferito dopo il rimbalzo, che provoca una temporanea variabilità sia delle posizioni che delle velocità e delle relative deviazioni standard.

La modificazione dei parametri di volo permette all'utente anche di creare ulteriori comportamenti collettivi più o meno caotici che tuttavia non rispecchiano il movimento desiderato nella simulazione. Nonostante con i valori impostati di default si sia cercato di parametrizzare delle interazioni che danno luogo a comportamenti emergenti analoghi allo spostamento di uno stormo, questo non è riuscito alla perfezione, e probabilmente vi sono combinazioni di valori che realizzano un effetto migliore.

6 Links

Link alla repository di GitHub utilizzata durante la realizzazione del progetto.

<https://github.com/Evyal/boids>