Modern C++ Programming

Dreamchess++

Report del Progetto

Mattia Zorzan

25 ottobre 2021

Indice

1 Project description

Su suggerimento del docente è stato scelto di non eseguire il porting dell'intero codice.
Mi sono concentrato prevalentemente sul contenuto della cartella `src` di dreamchess riuscendo a portare interamente tutte le funzionalità non grafiche.
Dato il lavoro individuale ho puntato a ridurre e semplificare quanto più possibile la logica, implementando tutte le possibilità di mosse corrette, unica pecca la necessità di semplificare la condizione di termine della partita. Nel progetto originale questa era legata alla GUI, ho quindi ammorbidito i vincoli chiudendo la partita quando uno dei due Re viene catturato.
Il risultato di tutto questo è dunque un eseguibile che mostra a terminale una scacchiera e permette di inserire una mossa sotto forma di stringa. I vari pezzi sono stampati tramite i corrispondenti caratteri unicode.

2 Project design

Affrontare il porting di un progetto come dreamchess ha sicuramente richiesto delle scelte abbastanza radicali nel design, soprattutto a causa (o meglio, per merito) delle possibilità offerte da *C++17*.

2.1 Macros

È stato in primis deciso di accantonare completamente l'utilizzo delle *Macro*, largamente utilizzate nel progetto originale per cose come la specifica dei pezzi della scacchiera e di alcune funzioni riguardanti essi, trasformate e riportate nell'omonima classe. Altre invece sono state completamente rimosse in quanto riguardanti la GUI oppure lo stato della schacchiera, controllabile tramite semplici funzioni e membri booleani delle classi.
Unica eccezione fatta per `#pragma`, utilizzato come *include guard* nella forma `#pragma once`.

2.2 Computazione

Da sottlineare la quasi totalità di computazione a runtime per il progetto, rendendo praticamente impossibile l'utilizzo sensato ed efficace di *templates* o `constexpr`.

2.3 Enumerazioni

Come anticipato, ho preferito rimuovere completamente l'utilizzo di Macro, è stato quindi necessario convertire quanto espresso in quella forma in maniera più "moderna", nasce in questo modo l'enumerazione di **Piece**, rappresentata tramite *Flag Enum*, ogni pezzo può essere visto come un valore a *8 bit*, i due più significativi indicano il colore, metre gli altri la tipologia del pezzo. All'assenza di un pezzo in un quadrato nella scacchiera corrisponde il valore 0.

2.4 Struttura del dato

Nella rappresentazione della schacchiera si è passati dal "raw" array di C al più moderno e sicuro `std::array`, questo ha permesso alla classe **Board** di ereditare `std::array::const_iterator`, rendendo possibile un'iterazione sui vari quadrati in modo più sicuro usando un *range-based for loop*.
È stato, naturalmente, preferito l'utilizzo di *smart pointers* invece di "raw" pointers (dove necessari).

3 Challenges

Nel continuo evolvere delle scelte implementative da adottare, 3 sono state fondamentali nel produrre il risultato finale

Rappresentazione dei pezzi Come discusso nella sezione precedente, l'unità fondamentale del gioco ha richeisto, come ci si poteva immaginare, il maggior tempo di sviluppo prima di arrivare alla sua forma definitiva.
Parte infatti come semplice `enum`, contenente una codifica numerica dei vari pezzi e colori messi in OR tra loro, quanto di più semplice si possa pensare. La promozione a classe avviene con la necessità di conoscere il *tipo* o il *colore* del pezzo di un determinato quadrato della schacchiera, vengono quindi overloadati gli operatori per poter usare i metodi statici `Piece::color` e `Piece::type`.

Validità delle mosse Forse la funzionalità che ha richiesto più debug. Sicuramente migliorabile alla fine si è riportata quasi interamente l'implementazione originale, alleggerita dal punto di vista dei parametri, sfruttando convenientemente la visibilità dei membri delle varie classi e qualche miglioramento logico. Il porting a C++ inoltre ha permesso di spezzare l'originale `board.c/h` nei sucuramente più espressivi `Move.cpp/hpp` e `Board.cpp/hpp`, potendo quindi dividere la logica delle mosse dai vari check sullo stato della scacchiera.

Notazione Il progetto originale supportava ogni tipologia di notazione, ho preferito utilizzare la notazione FEN in quanto riportava l'intero stato della scacchiera e non solo la mossa eseguita. La **History** è infatti composta da una serie di *Step*, composti dall'ultima mossa eseguita e dalla FEN dello stato attuale della scacchiera, in modo che sia facilmente ricostruibile.
Altra variazione dal progetto iniziale è quello di eseguire l'inizializzazione della schacchiera tramite parsing di una stringa FEN, invece dell'hard-coding dei vari pezzi nell'array `squares` di **Board**, utilizzata nell'originale dreamchess.

Input delle mosse Così come c'è stata una scelta personale nel cambiare la notazione per rappresentare la scacchiera, esiste una scelta obbligata nel cambio di notazione per l'input delle mosse.
Il progetto originale si avvaleva di un'interfaccia grafica mentre il proting si basa solamente sul terminale, è stato quindi necessario scegliere una formattazione per le stringhe di input, ovvero `rankfile-rankfile`, anche visibile come *source-destination*.
Stesso principio si applica sulla fromattazione delle *promotion moves*, la stringa di input dovrà essere formattata come segue `rankfile-rankfile=piece`. Se in una *promotion move* non viene specificato alcun pezzo esso verrà trattato come una promozione a Regina.
Entrambe le tipologie di mosse sono controllate sintatticamente tramite espressioni regolari.

4 Static Analysis

Sono stati utilizzati 3 analizzatori statici nel corso dello sviluppo, in primis i 2 disponibili con `clang` e `gcc`, oltre a questi è stato utilizzato anche `SonarLint`. Tutte le analisi sono terminate senza alcun problema per la parte di codice da me scritta, alcuni problemi sono stati trovati con la struttura del codice utilizzata nella repo originale, questa è stata però mantenuta e adattata dove serviva.