



HBStats

Relazione Progetto Freedom
2015 – 2016

Scopo del progetto

Lo scopo di HBStats è di gestire in tempo reale una partita di pallamano, mostrando in ogni istante le statistiche di gioco relative ad ogni componente di entrambe le squadre in gioco. L'idea è di poter ampliare questa applicazione con un database a cui ogni società può accedere per poter mantenere delle statistiche annuali, ovvero per la durata di un campionato. In questo progetto HBStats si limita alla gestione di singole partite, con solo una classifica delle squadre mantenuta per il campionato annuale. Vi è inoltre la possibilità di esportare file PNG, per consentire di aggiornare i dati relativi alle statistiche in modo automatico durante lo streaming in rete della partita in atto. I dati vengono memorizzati in file xml a cui è stata modificata l'estensione principalmente per personalizzazione.

Una partita di pallamano vede due squadre affrontarsi in due tempi da 30 minuti ciascuno. Entrambe le squadre sono composte da massimo 14 giocatori a disposizione in panchina, di cui solo 7 (6 giocatori di campo e 1 portiere) si trovano effettivamente in campo, e 2 allenatori. Ogni membro della squadra, sia esso in campo o in panchina, può ricevere dagli arbitri (2 per partita) varie tipologie di sanzioni: ammonizione (cartellino giallo), esclusione temporanea (due minuti) e esclusione (cartellino rosso). Il cartellino giallo ha lo scopo di ammonire un comportamento poco corretto o poco sportivo; i due minuti, che rappresentano la frazione di tempo in cui il giocatore che li ha ricevuti, o quello che li sconta in caso vengano ricevuti da un allenatore, è costretto a rimanere fuori dal campo, obbligando la squadra a giocare con un uomo in meno, hanno lo scopo di punire un comportamento antisportivo o potenzialmente pericoloso per gli altri giocatori; il cartellino rosso punisce comportamenti pericolosi o gravemente scorretti, e, oltre che in modo diretto, viene assegnato in seguito a 3 esclusioni temporanee individuali per i giocatori, una per gli allenatori.

Descrizione classi

Gerarchia

La gerarchia richiesta dai requisiti è composta dalle classi Tesserato, base astratta, Arbitro, Allenatore e Giocatore ad altezza 1, tutte e concrete, ed infine Portiere, ad altezza 2, derivata da Giocatore.

Tesserato: classe base della gerarchia, astratta, dati il distruttore virtuale ed il metodo virtuale puro *reset()*. I costruttori sono protetti poiché non è possibile istanziare oggetti di questo tipo (oltre al fatto che la classe è astratta), infatti il tipo Tesserato rappresenta un concetto comune, ma non concreto: non esiste un tesserato che sia solo tale; esso sarà sicuramente un arbitro, un giocatore, un allenatore, o qualche altro ruolo implementabile successivamente. Questa classe memorizza i campi comuni a tutti i tesserati, ovvero i dati anagrafici, più un booleano *checked* necessario per l'interoperabilità di model e view nel Model-View design pattern offerto dalle classi Qt.

Allenatore: classe concreta, deriva pubblicamente da Tesserato. Questa classe rappresenta un allenatore, il quale durante una partita può solamente ricevere sanzioni. Il metodo *reset()* è implementato in modo che resettò le sanzioni ricevute da un allenatore nel corso di una partita.

Arbitro: classe concreta, deriva pubblicamente da Tesserato. Questa classe rappresenta un arbitro di pallamano. Essa memorizza il livello dell'arbitro, che determina quale categoria di partite egli può arbitrare ed è determinato dal numero di partite per categoria già arbitrate. Il livello è memorizzato come un intero positivo, da 0 a 3, dove 0 rappresenta il livello minimo, e quindi la possibilità di arbitrare solamente partite di categoria regionale, mentre 3 rappresenta il livello massimo, e quindi la possibilità di arbitrare ogni categoria, compresa quella internazionale. Il livello rappresenta oltre ad un dato reale (più specifico nella realtà, che prevede 6 tipi di qualifiche, conseguite tramite corsi ed esami appositi), anche l'esperienza in termini di partite arbitrate. In questa classe il metodo *reset()* viene implementato per non fare nulla, mentre si può procedere alla modifica del livello e del numero di partite arbitrate per categoria tramite appositi metodi.

Giocatore: classe concreta, deriva pubblicamente da Tesserato. Questa classe rappresenta un giocatore in una partita di pallamano. Esso può segnare o meno durante una partita, rigori dai 7 metri o tiri "normali". Questi sono quindi memorizzati per consentire poi il calcolo delle statistiche personali e le percentuali di realizzazione in una partita. La classe Giocatore implementa il metodo *reset()* in modo tale che una chiamata *g.reset()* azzeri i valori dei tiri e delle sanzioni del Giocatore *g*.

Portiere: classe concreta, deriva pubblicamente da Giocatore. Questa classe rappresenta un portiere in una partita di pallamano, e deriva da Giocatore in quanto ad un portiere è concesso giocare come giocatore di campo, anche se praticamente questo avviene raramente. Di un portiere si memorizzano le parate avvenute con successo, relativamente a tiri e rigori effettuati dalla squadra avversaria. Questi elementi servono inoltre per calcolare le percentuali di "efficienza", un dato molto importante nel corso di una partita, che in pratica determina la vittoria o meno della squadra: un portiere con una buona percentuale assicura più contrattacchi (detti contropiedi), più difficili da gestire per una difesa, e dunque più realizzativi. Il metodo *reset()* azzerà questi dati.

Altre classi logiche

Vettore: la classe vettore rappresenta il contenitore richiesto nei requisiti. Viene implementato un array dinamico, basato sullo *std::vector*, con i principali metodi per inserimento, modifica e rimozione degli elementi memorizzati. Il metodo statico *T* ridimensiona(Vettore<T> & v)* si occupa del ridimensionamento dinamico, copiando gli elementi di *v* in un nuovo array con capacità aumentata di un fattore costante di default, *DEFAULT_DIMENSION*, settato a 10. Questa scelta è stata fatta in base all'utilizzo del contenitore, ovvero per memorizzare elementi polimorfi della gerarchia, limitando al minimo lo “spreco” di memoria allocata sullo heap. Questa scelta verrà discussa successivamente.

Iteratore: questa classe rappresenta un iteratore. La scelta di creare questa classe indipendentemente come template è stata fatta per non appesantire la classe vettore e per poter definire una volta sola iteratori costanti e non. Questo avviene tramite la struttura

```
template <bool flag, class _nConst, class _Const>
struct const_or_notc;
```

Questa struttura viene poi ridefinita nelle linee seguenti nelle versioni con *flag* uguale a *true* e con *flag* uguale a *false*.

```
template <class _nConst, class _Const>
struct const_or_notc<true, _nConst, _Const>{
    typedef _Const type;
};

template <class _nConst, class _Const>
struct const_or_notc<false, _nConst, _Const>{
    typedef _nConst type;
};
```

La prima versione definisce i tipi costanti, mentre la seconda quelli non costanti.

In questo modo nella definizione dei tipi *pointer* e *reference* di un *template<T, is_const>* *Iteratore* avviene in modo “automatico” in forma costante o meno a seconda del parametro *is_const*.

Vengono poi definiti vari metodi caratteristici degli iteratori.

Squadra: questa classe rappresenta un insieme di giocatori e allenatori. Viene utilizzato il contenitore Vettore per memorizzare oggetti polimorfi della gerarchia, ovvero un insieme di tesserati. Essa memorizza i principali dati necessari al mantenimento di una classifica e offre metodi per calcolare i punti, la differenza reti, i goal segnati dai giocatori in una partita, e per aggiungere, modificare e rimuovere tesserati; il metodo *reset()* resetta tutti i tesserati che compongono la squadra, mentre il metodo *clear()* svuota il vettore e dealloca i tesserati. Viene fornito, inoltre, l'override dei principali metodi di accesso al vettore.

La scelta di implementare e utilizzare un vettore per memorizzare oggetti di tipo Tesserato è basata sulla quantità di accessi casuali e sequenziali che avviene in HBStats. Inoltre la dimensione (size) di un Vettore molto raramente supera le 30 unità, e per questi numeri un vettore dinamico garantisce maggiore velocità. Gli unici svantaggi si hanno nell'inserimento, nella rimozione e nell'ordinamento, considerando che il ridimensionamento, avendo impostato la dimensione di default a 10, questo avviene all'inserimento del primo elemento e un altro paio di volte.

SquadreModel: deriva pubblicamente da QAbstractListModel, per poter interagire con le views in modo immediato. Vengono reimplementati i principali metodi consigliati nella documentazione e necessari per il corretto funzionamento, più vari metodi per inserire,

modificare e rimuovere squadre e tesserati, ordinare le squadre in base ai punti, gli operatori per accedere alla lista delle squadre. È stato scelto di utilizzare una QList per memorizzare puntatori a oggetti Squadra allocati sullo heap, in quanto le operazioni principali effettuano inserimenti nel mezzo o ordinamento, mentre gli accessi sono quasi sempre sequenziali, raramente casuali. La scelta di derivare la classe da QAbstractListModel è invece stata effettuata in quanto nel progetto come views vengono utilizzate principalmente QListView.

ArbitriModel: come SquadreModel, anche ArbitriModel deriva da QAbstractListModel, per gli stessi motivi. Allo stesso modo vengono reimplementate le classi necessarie all'interazione tra model e view, più vari metodi per l'inserimento, modifica e rimozione di arbitri.

CheckList: deriva pubblicamente da QStringListModel. L'implementazione di questa classe si è resa necessaria per la creazione di una lista con elementi aventi una checkbox per selezionarli. Porre un valore bool nel costruttore di questa classe ha fatto sì che questa classe fruisse da modello sia per una lista di tesserati, memorizzati in una QStringList con le principali informazioni identificative, selezionabili, sia come lista semplice.

XmlHandler: questa classe ha lo scopo di gestire la scrittura e la lettura del database xml (salvato con l'estensione “.hbs”). Opera tramite le classi messe a disposizione dalla libreria Qt.

Eccezioni: vengono definite varie classi di eccezioni. Err_Ammonizione, Err_DueMinuti ed Err_Escusione vengono lanciate in caso di violazione dei limiti relativi alle sanzioni per giocatori e allenatori, Err_Tesserato segnala la presenza di un tesserato identico già presente, mentre Err_Open e Err_Save servono a gestire le operazioni di apertura e salvataggio.

Classi grafiche

MainWindow:

NewWizard:

IntroPage:

PersonaPage:

SquadraPage:

PartitaPage:

Editor:

PushButton:

Tabs:

LinePartita:

Partita:

LineStat:

Stat:

Descrizione dell'uso di codice polimorfo

Manuale utente

Specifiche