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 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 resetti 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()* azzera 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 la 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 re-implementati 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 QAstractListModel è invece stata effettuata in quanto nel progetto come views vengono utilizzate principalmente QListView e QComboBox.

ArbitriModel: come SquadreModel, anche ArbitriModel deriva da QAstractListModel, per gli stessi motivi. Allo stesso modo vengono re-implementate 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_Esclusione 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: la finestra principale, crea e controlla l'interazione dell'utente con i vari menù, deriva da QMainWindow, cosicché venga sfruttato il distruttore offerto dalla libreria Qt.

NewWizard: questa classe deriva pubblicamente da QWizard e rappresenta una procedura guidata per la creazione di nuovi elementi, siano essi tesserati, squadre o partite. Viene reimplementato il metodo *accept()* per consentire un controllo delle informazioni immesse nella creazione di una partita. La creazione dei layout delle pagine è affidata alle classi seguenti.

IntroPage: è la pagina iniziale di NewWizard, permette di selezionare l'elemento che si vuole creare.

PersonaPage: questa pagina si occupa di creare tesserati. In essa è presente una QComboBox in cui viene utilizzata la classe SquadreModel come modello.

SquadraPage: questa pagina si occupa di creare una o più squadre.

PartitaPage: permette di creare una partita, selezionando le squadre, i giocatori e gli arbitri tra quelli già esistenti. Per poter accedere a questa pagina sono necessari almeno 2 arbitri e due squadre in memoria, mentre per poter procedere alla creazione effettiva di una partita è necessario selezionare due persone distinte nei campi destinati agli arbitri, due squadre distinte nei campi squadra e un numero di giocatori compreso tra 7 e 14 per squadra. Inoltre la categoria deve essere compatibile con il livello degli arbitri. In questa pagina vi sono due QComboBox con modello SquadreModel, due con modello ArbitriModel e due QListView con modello CheckList, settata sulla base della squadra corrente, con il flag *checkable* posto a *true*.

Editor: deriva pubblicamente da QDialog e consiste in un form per la modifica di alcuni campi dei tesserati e delle squadre create, oltre che per l'eliminazione degli stessi. Una QListView si occupa di mostrare all'utente la lista degli elementi che si possono modificare, cambiando model a seconda della scelta fatta sui QRadioButton e, in caso si voglia modificare un tesserato di una squadra, su una QComboBox, il cui modello è SquadreModel.

PushButton: deriva pubblicamente da QPushButton, ed si è reso necessario creare questa classe per definire alcuni segnali derivanti dall'utilizzo del mouse da parte di un utente. In particolar modo è stato definito un segnale *rightclicked* e i *doubleclicked* rispettivamente ai tasti sinistro e destro del mouse.

Tabs: deriva da QTabWidget e viene creata in seguito alla creazione di una partita. Consiste di 3 pagine, Partita due Stat. Controlla l'interazione tra le varie pagine e implementa uno slot fine all'esportazione in formato PNG delle pagine Stat.

LinePartita: rappresenta un tesserato di una squadra nella pagina Partita, con tutti i vari PushButton associati. Connette i segnali alle funzionalità offerte dal tesserato stesso.

Partita: una pagina partita è costituita da tante LinePartita quanti sono i tesserati "convocati" di entrambe le squadre. Si occupa della gestione dei portieri, consentendo una conversione nel caso il giocatore selezionato come portiere corrente non fosse stato precedentemente memorizzato come tale. In questa conversione viene evitato lo *static_cast*, utilizzando invece il costruttore di copia opportunamente definito nella classe Portiere.

LineStat: rappresenta un tesserato nella pagina Stat, con le varie statistiche offerte dai metodi del tesserato stesso.

Stat: questa pagina rappresenta le statistiche di una squadra per la partita corrente.

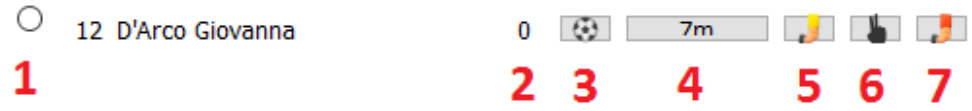
Descrizione dell'uso di codice polimorfo

Nella gerarchia sono stati resi virtuali i metodi *getInfo(), reset(), modifica(),* e il distruttore nella classe Tesserato. Questo agevola varie operazioni nella classe Squadra, dove in particolare la modifica dei tesserati viene effettuata direttamente senza preoccuparsi di individuare il tipo dinamico dei tesserati in oggetto. Anche in Editor la modifica del tesserato selezionato avviene sulla base del tipo dinamico del tesserato per quanto riguarda la modifica di una persona in una squadra, anche se è necessario conoscerne il tipo per i campi specifici del tesserato selezionato, che altrimenti non potrebbero venire memorizzati. In LinePartita e in Partita vengono effettuati vari *dynamic_cast* per abilitare o disabilitare PushButton, mentre per aggiornare i dati si sfruttano i metodi virtuali dedicati per quanto riguarda le sanzioni, mentre per aggiornare i dati specifici della classe Giocatore si utilizza un dynamic_cast per un test condizionale. Per quanto riguarda gli operatori di confronto, essi vengono dichiarati virtuali nella classe base Tesserato, e ne viene fatto l'override solamente nella classe Giocatore. In questa classe viene fatto controllato il tipo dinamico del parametro passato come riferimento costante, tramite un dynamic_cast per consentire l'uguaglianza anche tra oggetti della classe Giocatore e oggetti della sottoclasse Portiere, in questo caso (la classe Giocatore potrà essere ampliata inserendo sottoclassi specifiche come Terzino, Centrale, Ala, Pivot, che identificano i vari ruoli in campo di un giocatore diverso dal portiere). È stato scelto infatti di poter eseguire il confronto anche tra sottoclassi diverse, richiamando l'operatore di uguaglianza della classe base, ed evitando in tal caso i confronti tra caratteristiche specifiche delle sottoclassi, senza dover però identificare il tipo dinamico dei due elementi prima dell'utilizzo dell'operando stesso.

Si è scelto di rendere virtuali (puri nella classe base) i metodi relativi alle sanzioni, per evitare ripetitivi controlli tramite dynamic_cast, sebbene la classe Arbitro non necessiti di questi metodi. Essendo necessario implementare tali metodi nelle sottoclassi, vengono implementati in Arbitro in modo da non fare nulla o lanciare eccezioni.

Manuale utente

L'interfaccia di HBStats è abbastanza semplice e intuitiva, tranne che per la pagina Partita di Tabs.



1 – Permette di selezionare il portiere in campo, ovvero il giocatore che si trova al momento tra i pali della porta. È possibile selezionarne solo uno per volta per squadra, e nel caso il giocatore selezionato non sia salvato come Portiere, verrà chiesta conferma per una conversione: se confermata il giocatore selezionato sarà convertito, altrimenti la selezione non avverrà e tornerà ad essere portiere corrente il portiere che era già selezionato.

2 – Indica i goal effettuati dal giocatore.

3 – Permette di aggiungere o rimuovere tiri segnati o sbagliati come segue:

a) **Click singolo sinistro**: aggiunge un tiro segnato. Il contatore (2) aumenta di uno.
b) **Doppio click sinistro**: aggiunge un tiro sbagliato. Il contatore (2) mostrerà in rapida successione un aumento e un decremento, tornando al valore precedente.
c) **Click singolo destro**: rimuove un tiro segnato. Il contatore (2) decrementa di uno.
d) **Doppio click destro**: rimuove un tiro sbagliato. Viene visualizzato un messaggio di avvenuta modifica.

4 – Permette di aggiungere o rimuovere rigori segnati o sbagliati. Funziona come per i tiri. Si noti che aggiungendo un rigore, verrà in automatico aggiunto un tiro nella pagina delle statistiche.

5 – Permette di aggiungere o rimuovere una ammonizione come segue:

a) **Click singolo sinistro**: ammonisce il giocatore. Il pulsante diventa giallo a segnalare che il tesserato è già ammonito. Se si tenta di aggiungere un'altra ammonizione, verrà visualizzato un messaggio di errore.
b) **Click singolo destro**: rimuove l'ammonizione se il giocatore è ammonito, altrimenti non fa nulla.

6 – Permette di aggiungere o rimuovere un'esclusione temporanea come segue:

a) **Click singolo sinistro**: aggiunge un'esclusione temporanea, o visualizza un messaggio di errore nel caso sia già stato raggiunto il numero massimo di esclusioni temporanee per il tesserato in questione.
b) **Click singolo destro**: rimuove un'esclusione temporanea.

7 – Permette di aggiungere o rimuovere un'esclusione come per le ammonizioni. Il pulsante in caso il tesserato sia escluso è colorato di rosso.

Per quanto concerne le azioni della toolbar relative a una partita:

Reset: azzera tutti i contatori relativi a tiri e sanzioni per tutti i tesserati di entrambe le squadre.

Termina: termina la partita, disabilita la pagina Partita e salva il risultato in memoria, aggiungendo una vittoria, pareggio o sconfitta a seconda del risultato ad entrambe le squadre.

Chiudi: chiude le tabs senza salvare il risultato e ritorna alla schermata principale.

Specifiche

Il progetto è stato sviluppato su un sistema Ubuntu versioni dalla 15.10 fino a 16.04 LTS (il sistema viene mantenuto aggiornato). L'IDE utilizzato è QtCreator 3.2.1, basato sulla libreria Qt 5.3.2, e compilatore GCC 4.6.1 64 bit. Lo stesso progetto è stato testato su sistema Windows 10, tramite il tool QtCreator e nelle macchine del laboratorio tramite terminale, richiamando preventivamente il file qt-5.3.2.sh per impostare la corretta versione di Qt e qmake. Non sono stati riscontrati errori e tutto compilava ed eseguiva correttamente.
Viene fornito il file *progetto.pro* in quanto è stato necessario aggiungere le righe "*QT += core gui*" e "*greaterThan(QT_MAJOR_VERSION, 4): QT += widgets*", altrimenti non incluse nel file generato dalla modalità progetto di qmake, ovvero dal comando *qmake –project && qmake*.
Tra i file consegnati è presente il file *risorse.qrc* necessario per la corretta visualizzazione delle immagini, che risiedono dentro la cartella *images*, e un file *esempio.hbs*, contenente un semplice database esempio.