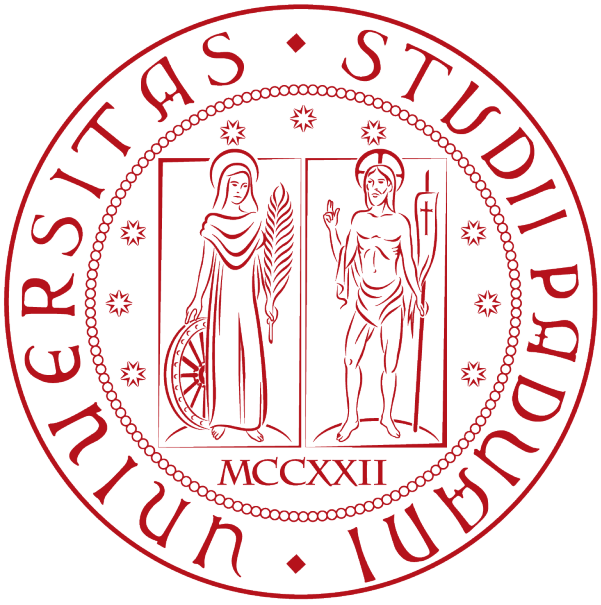

Relazione PAO

Mattia Bano - 20 Giugno 2017

Tipologia Utente	Password
Admin	admin
Moderatore	moderatore
UtentePro	utentepro
UtenteFree	utentefree

Indice

1 Scopo del progetto

Lo scopo del progetto è realizzare un'applicazione in C++ utilizzando la libreria Qt per la costruzione di una GUI.
Il progetto è stato pensato come un database di carte di un ipotetico gioco chiamato "MinionStone" che prende spunto dal popolare gioco online "HearthStone".
Le carte sono di due tipi principali, Magia o Creatura. Le Creature si dividono ulteriormente in sottotipi come Bestia, Drago, Pirata, ecc...
L'applicazione salva e carica dati in e da un file XML in modo trasparente rispetto all'utente.
Gli utenti del sistema si dividono in 4 tipi, Admin, Moderatore, UtentePro e UtenteFree. Ogni tipologia di utente ha diversi permessi, gli Admin hanno il controllo completo dell'applicazione, i Moderatori hanno il controllo completo sulle azioni riguardanti le carte, gli UtentiPro possono modificare le carte mentre gli UtentiFree possono solo consultare l'elenco.
Nel progetto è stato utilizzato il pattern MVC cercando di mantenere netta la divisione tra logica e presentazione grazie ad un modulo di controllo.

2 Gerarchie di tipi

Il progetto presenta cinque gerarchie, due nella parte logica e tre nella parte grafica.

2.1 Gerarchia di carte

La prima gerarchia rappresenta l'insieme delle carte visualizzate nel programma. Si parte con una classe base astratta "Carta" che rappresenta una generica carta da gioco con i seguenti metodi virtuali puri:

- `costoPolvereBase()` restituisce il costo base in polvere di una carta (la polvere è una moneta virtuale all'interno del gioco);
- `costoPolvere()` calcola in vari modi il costo in polvere delle carte partendo dal costo base;
- `getTipoCarta()` restituisce una stringa contenente il tipo della carta.

Dalla classe base derivano due classi, "Magia" e "Creatura" la prima concreta e la seconda astratta. La prima rappresenta una carta di tipo magia che implementa i metodi virtuali puri. La seconda rappresenta una generica carta creatura a cui sono stati aggiunti due campi dati utilizzati in altrettanti metodi e cinque metodi virtuali puri:

- `attaccoTotale()` calcola l'attacco totale della creatura a partire dal suo attacco di base;
- `vitaTotale()` calcola la vita totale della creatura a partire dalla sua vita di base;
- `typeMultiply()` restituisce il moltiplicatore per il calcolo del costo in polvere di una carta creatura;
- `getTipoCreatura()` restituisce una stringa contenente il tipo della creatura;
- `getEffettoBase()` restituisce l'effetto base permanente di una carta creatura.

Dalla classe "Creatura" derivano sei classi che implementano tutti i metodi virtuali e rappresentano i tipi di creatura esistenti all'interno del gioco.

Questa gerarchia è facile da ampliare con nuovi tipi di carta e creatura o con ulteriori tipi di carte astratte da cui derivare tipi più complessi, ad esempio si possono aggiungere armi o magie di diverso tipo.

2.2 Gerarchia di utenti

La gerarchia di utenti rappresenta l'insieme degli utenti che possono utilizzare il sistema. La classe base "Utente" è astratta e da essa derivano due classi concrete, "UtenteFree" e "UtentePro" e una classe astratta "Amministratore". Le classi concrete rappresentano le due tipologie di utenti con meno permessi. La classe

astratta rappresenta la classe base per i due tipi concreti, "Admin" e "Moderatore", che rappresentano le tipologia di utenti con più permessi. In questa gerarchia è presente un metodo virtuale puro `getTipo()` che restituisce una stringa con il tipo di utente.

2.3 Gerarchia presentazione model

La GUI principale dell'applicazione si differenzia a seconda del tipo di utente che vi accede. Si è pensato di creare una gerarchia partendo da una lista base aggiugendo funzionalità man mano che si scende nella gerarchia. In questo modo si evita il riutilizzo di codice per GUI molto simili.

2.4 Gerarchia per la scelta della creazione di utente/carta

Questa gerarchia è composta da una classe base astratta e due classi concrete. La classe base contiene un metodo virtuale puro `setVariables()` utilizzato per settare le label dei bottoni e della richiesta di input da parte dell'utente.

2.5 Gerarchia per la creazione/modifica delle carte/utenti

L'ultima gerarchia per la parte grafica è utilizzata per la creazione e la modifica di carte e utenti. La gerarchia è composta da una classe base astratta dalla quale derivano due classi concrete, da cui a loro volta derivano quattro classi concrete. La gerarchia è utilizzata per creare delle GUI molto simili ma che si differenziano in alcuni comportamenti, in questo modo si evita di creare classi molto simili tra di loro. Il metodo virtuale puro alla base della gerarchia è `setFormLayout()`, utilizzato per creare la form all'interno della GUI, che cambierà valore in base alla classe istanziata.

3 Uso del polimorfismo

Il primo e più lampante uso di codice polimorfo è visibile nei distruttori virtuali presenti alla base di tutte le gerarchie presenti nel progetto.

3.1 Gerarchia di carte

All'interno della gerarchia di carte troviamo otto metodi virtuali:

- `costoPolvere()`, `getTipoCarta()`, `attaccoTotale()`, `vitaTotale()` e `getTipoCreatura()` vengono utilizzate dal controller (righe 159-173) nella creazione delle entry della tabella;
- `costoPolvereBase()` viene utilizzato all'interno delle invocazioni di `costoPolvere()` come base per il calcolo della prima;
- `typeMultiply()` viene utilizzata all'interno delle invocazioni di `costoPolvere()` nel calcolo della polvere per le carte creatura;
- `getEffettoBase()` restituisce l'effetto base delle carte creatura; non è utilizzato all'interno del progetto, è stato comunque lasciato perchè utile all'implementazione di funzionalità future.

3.2 Gerarchia di utenti

Nella gerarchia di utenti esiste un solo metodo virtuale puro `getTipo()` utilizzato nella creazione delle entry per la tabella utenti (riga 196), nella crezione della finestra di modifica utenti (righe 483 e 500), per la modifica dell'oggetto utente da modificare dopo che l'utente ha inserito i dati in modo corretto (righe 582 e 633) ed infine viene utilizzato nella classe "UserWriter" per scrivere i dati dal contenitore al file xml corrispondente (righe 30, 35, e 45).

3.3 Gerarchia per la scelta utente/carta

In questa gerarchia (che parte dalla classe base "ChooseDialog") è presente un metodo virtuale puro utilizzato all'interno dei costruttori delle classi concrete per settare le label della richiesta all'utente e le label dei bottoni per selezionare la scelta (righe 5 in "ChooseCard" e in "ChooseUser").

3.4 Gerarchia per la creazione/modifica di carte

Nell'ultima gerarchia che contiene polimorfismo, il metodo virtuale puro `setFormLayout()` è utilizzato all'interno dei costruttori per costruire la form giusta in base alla view di creazione/modifica costruita (riga 6 in tutte le classi concrete derivate dalla classe base "CreateView").

4 Manuale utente

L'applicazione inizia con il caricamento, trasparente per l'utente, dei file in cui sono inseriti il database utenti e il database carte. Se i due file non sono presenti, l'applicazione crea un utente con username e password "admin". La GUI principale, successiva al login, è una tabella con possibilità di creare, modificare ed eliminare carte (in base ai propri permessi) e se si è admin anche di visualizzare gli utenti del sistema e di fare le stesse operazioni su di essi. Per creare una carta/utente, viene chiesto che tipo di carta/utente creare, successivamente alla scelta si viene portati alla finestra di creazione in cui si devono inserire dati con alcune limitazioni per la creazione di utenti:

- lo username può contenere solo lettere, numeri e deve essere compreso tra tre e sedici caratteri;
- la password può contenere solo lettere, numeri e deve essere compreso tra otto e sedici caratteri;
- l'e-mail deve avere il classico formato delle e-mail;
- il nome e il cognome possono contenere solo lettere e sono limitati tra tre e sedici caratteri.

Per modificare/eliminare una carta/utente è necessario selezionare la riga da modificare/eliminare e poi cliccare il bottone corrispondente. L'utente Admin ha anche la possibilità di passare dalla visualizzazione di carte a quella di utenti e viceversa grazie a due radio button. Infine, il bottone "logout" permette di uscire dal profilo con cui si è acceduto e rientrare con un'altro profilo.

A Indicazioni conclusive

A.1 Impegno temporale

- progettazione modello: 5h
- progettazione GUI: 3h
- codifica modello: 10h
- codifica GUI: 15h
- codifica controller: 30h
- debugging: 5h
- test: 2h

Il monte di 60 ore è stato superato a causa delle difficoltà riscontrate nel codificare la classe "Controller" in quanto si voleva utilizzare il modello MVC con la massima separazione tra Model e View.

A.2 Informazioni tecniche

- Sistema operativo: Windows 10 Education 64-bit
- Compilatore: MiniGW 5.3.0
- Libreria QT: 5.7.0
- QT-Creator: 4.0.2

A.3 Compilazione

Per compilare con i computer del laboratorio, posizionarsi da terminale all'interno della cartella "MinionP2", eseguire i comandi "qmake" e "make". Queste operazioni generano un file eseguibile che potrà essere avviato con il comando "./MinionP2".