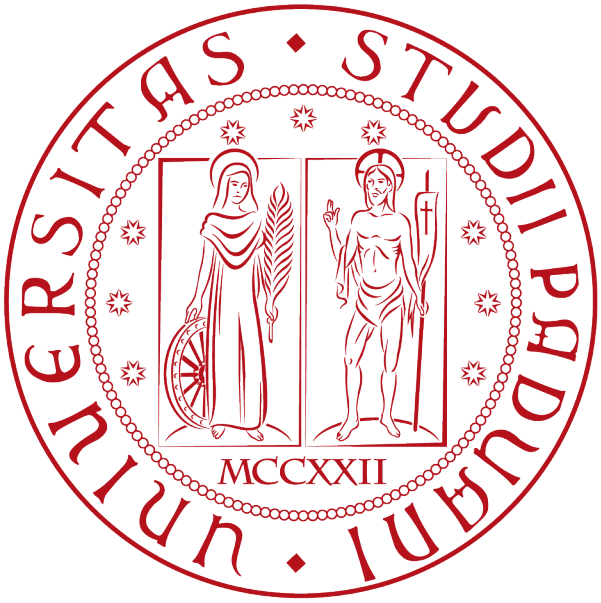

Relazione Prog. ad Oggetti

Japp - Games & Consoles

Giulia Amato, matricola 1075626 - Anno 2016/2017

Indice

1 Scopo del progetto

Il progetto si prefigge lo scopo di realizzare un programma che simuli un Database collaborativo dedicato all'ambito videoludico, concedendo agli utenti specifici permessi. Gli utenti si dividono in Admin, User Base e User Premium. In particolare si vogliono fornire le seguenti funzionalità:

1. **Admin**: aggiunta, rimozione e modifica dei prodotti e degli utenti. La possibilità di creazione degli utenti è riservata esclusivamente all'amministratore, che potrà così definire quali utenti possono dare un contributo, e quali invece possono solo avere una funzione consultativa.
2. **User Base**: consultazione del database (prodotti e utenti) e ricerca semplice tramite stringa.
3. **User Premium**: funzionalità dell'user base, con aggiunta: servizi di interrogazione specifici che mirano alla visualizzazione di prodotti compatibili con quanto cercato, aggiunta, modifica ed eliminazione dei prodotti.

1.1 Accesso

Accesso Amministratore:
Nickname: **admin** Password: **admin**
Accesso User base:
Nickname: **DT** Password: **imcool**
Accesso User Premium:
Nickname **giulia** Password: **giulia**

1.2 Compilazione ed esecuzione

Per compilare il progetto è necessario eseguire il comando `qmake ProgettoP2.pro`, contenuto all'interno della cartella ProgettoP2, tale file permetterà la generazione automatica tramite qmake del Makefile, e successivamente lanciare il comando `make`.
Per l'esecuzione invece: `./ProgettoP2`
Vengono consegnati anche i file productDatabase.xml e userDatabase.xml come esempi di dati. Entrambi i file si possono trovare nella cartella model. Se entrambi i file non dovessero essere presenti, l'applicazione verrà eseguita ugualmente e in tal caso verrà creato di default un utente amministratore con nickname `admin` e password `admin`.

2 Struttura

2.1 Contenitore

La scelta del contenitore è ricaduta sulla struttura QList<T>, un template di classe presente nella libreria di Qt che rappresenta le liste. La preferenza di una lista è dovuta al fatto che, data la natura di Database dell'applicazione, è necessario che le classiche operazioni di inserimento e cancellazione siano più rapide possibili, in questo caso dunque in tempo O(1). Ho usato QList<T> sia per il database dei prodotti (productdatabase), in cui avrò una semplice lista di puntatori a prodotto, sia per il database degli utenti (userdatabase), in cui avrò una lista di puntatori ad utente. Per quanto riguarda la gestione della memoria sono stati definiti dei metodi di rimozione.

2.1.1 Pattern Model-View-Controller

Per lo sviluppo si è cercato di seguire quanto più possibile una scelta architetturale del pattern MVC, così da separare l'implementazione logica da quella grafica. Nel Model, infatti, vengono gestiti direttamente i dati; nella View vengono visualizzati i dati contenuti nel model fornendo una rappresentazione grafica, occupandosi dell'interazione con gli utenti, così come semplici controlli di inserimento dati; nel Controller invece viene definita la logica di controllo.

3 Gerarchie di tipi

In seguito illustro le principali scelte progettuali effettuate per la realizzazione del modello dei dati del programma:

3.1 Gerarchia dei prodotti

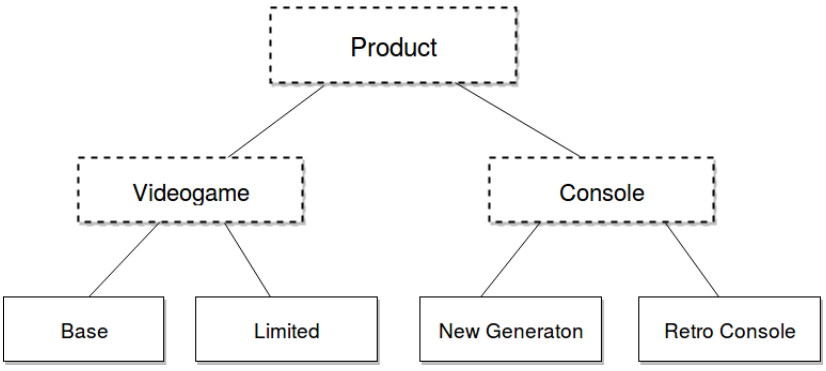

Figura 1: Gerarchia prodotti

3.1.1 Classe base astratta Prodotto

Tale gerarchia costituisce l'insieme dei prodotti raccolti nel contenitore C. Si parte da un generico prodotto nella classe base astratta **Product**, che è alla base della gerarchia. I campi dati sono:

- `nome` : Nome del prodotto.
- `sviluppatore` : Nome dello Sviluppatore / Casa di Sviluppo.
- `data` : Data di inserimento nel Database.

Un prodotto viene identificato univocamente da un valore static int che verrà incrementato ogni volta che verrà creato un nuovo oggetto. L'astrazione di tale classe è data dai seguenti metodi virtuali:

- `virtual QString getTypeProd() const=0;` : metodo virtuale puro che restituisce una QString contenente il tipo del prodotto, utilizzato al fine di non usare eccessivamente il dynamic-cast.
- `virtual void saveProduct(QXmlStreamWriter &)=0;` : metodo virtuale puro che si occupa del salvataggio dei dati del prodotto, esso verrà implementato in ogni sottoclasse.
- `virtual void loadProduct(QXmlStreamReader &)=0;` : metodo virtuale puro che si occupa del caricamento dei dati del prodotto, esso verrà implementato in ogni sottoclasse.

Successivamente vengono definite le due classi astratte **Videogame** e **Console** derivate da Product, che definiscono e distinguono la natura dei prodotti che verranno concretizzati.
I campi dati comuni di ogni prodotto vengono caricati con il metodo
`void loadCommonFieldsProduct(QXmlStreamReader&)`e scritti nel file .xml con il metodo
`void saveCommonFieldsProduct(QXmlStreamWriter&)`.

3.1.2 Classe astratta Videogame

La classe **Videogame** ha come campi dati:

- `PEGI (Pan European Game Information)` : sistema di classificazione in base all'età del giocatore.
- `platform` : la piattaforma console sulla quale far girare un gioco (NOTA: Nel DB può essere presente il medesimo videogioco ma in diverse piattaforme).
- `genre` : genere di appartenenza del videogioco.
- `remake`: campo dati utile per segnalare se un videogioco è un adattamento di un titolo uscito precedentemente con l'intento di modernizzare il gioco per un nuovo hardware.

3.1.3 Classe Base e Limited

La classe Videogame a sua volta si concretizza nei prodotti **Base** e **Limited**; un videogame Base è un entità digitale e perciò viene ulteriormente estesa di informazioni dal campo `Dimensione` che descrive la dimensione del file, mentre un videogame Limited rappresenta una particolare versione fisica di un Videogame. Una Limited, essendo un'edizione limitata, ha dei contenuti speciali oltre al gioco base:

- `steelbook` : indica se la copertina è in versione standard o in versione steelbook.
- `DLC` : indica se ci sono contenuti scaricabili aggiuntivi in bundle.
- `actionFigure` : indica se nel pacchetto è incluso un modellino.
- `soundtrack` : indica se nel pacchetto è inclusa una colonna sonora.
- `extraContents` : indica se in bundle ci sono altri contenuti oltre a quelli già citati in precedenza.
- `artbookpages` : indica il numero di pagine dell'Artbook del videogioco.

3.1.4 Classe astratta Console

La classe astratta **Console**, che rappresenta una generica console presente nel database è stata implementata per permettere una maggiore varietà della natura degli oggetti memorizzati. Troviamo come campi dati:

- `controllerTechType` : indentifica la tipologia della tecnologia usata dal controller (se cablato o wireless).
- `hardDisk` : esprime la capacità del disco.
- `maxResolution` : identifica la massima risoluzione video ottenibile.
- `consoleColor` : rappresenta la colorazione della scocca della console.
- `specialEdition` : è un booleano che indica se la console è una versione speciale (tiratura limitata).

3.1.5 Classe NewGeneration e RetroConsole

Tale classe astratta viene concretizzata nelle classi **NewGeneration** e **RetroConsole**; la prima rappresenta una console di nuova generazione, perciò il suo unico campo dati è `support4K` che è una proprietà specifica atta a segnalare il supporto all'Ultra HD. Per quanto riguarda invece **RetroConsole**, che rappresenta una console di vecchia generazione, l'unico campo dati è `bit,` campo utile a differenziare le RetroConsole per generazione (8 bit, Terza Generazione, anni 1983-1995), (16 bit, Quarta Generazione, 1988 - 1999), (32 bit, Quinta Generazione, 1993 – 2006). Ogni classe presenta un metodo virtuale `getTypeProd()` che restituisce una stringa contenente l'appartenenza ad una specifica categoria di prodotto, inoltre presenta le funzioni virtuali di `Save` e `Load` per facilitare la lettura e la scrittura su file formato xml.

3.2 Gerarchia degli utenti

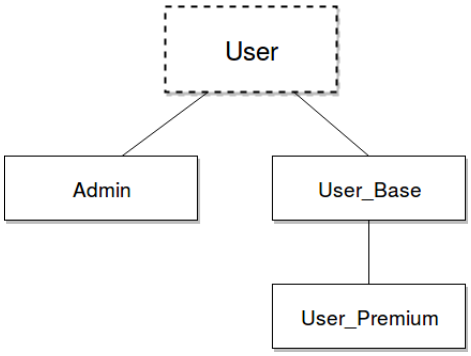

Figura 2: Gerarchia Utenti

La gerarchia utenti consiste di una classe astratta **User**, resa astratta dai metodi virtuali puri `getLabel()`, che restituisce una stringa contenente l'informazione circa l'appartenenza ad una data categoria d'utenza; mentre `ModifyUsers()`, `ModifyProducts()` e `premiumFunctions()` permettono la differenziazione di permessi specifici che serviranno per le View.

3.2.1 Classe base astratta User

Per quanto riguarda la caratterizzazione di un generico utente si è deciso di definire i seguenti campi dati di tipo QString:

- `nickname` : Un utente viene identificato univocamente dal suo nickname
- `password` : Password dell'utente

- `name` : Nome dell'utente
- `surname` : Cognome dell'utente

Tale classe è resa astratta dai metodi virtuali puri `virtual QString getLabel() const =0` che nelle classi derivate restituisce una QString che identifica la tipologia di appartenenza dell'utente; `virtual bool ModifyUsers() const =0`, `virtual bool ModifyProducts() const =0` e `virtual bool premiumFunctioncs() const =0` sono dei metodi che una volta implementati restituiscono dei valori booleani che permettono la diversificazione delle View.

3.2.2 Classe User_Base, User_Premium e User_Admin

Tali classi non aggiungono campi dati ulteriori a quelli già ereditati, ma implementano i metodi virtuali già citati con le dovute differenziazioni di permessi. La scelta di diversificare le view per tipologia di utenza è stata definita marcando cosa si aspetterebbe di fare un dato utente. Ad esempio l'amministratore non ha la stessa possibilità di ricerca mirata che è stata creata per l'user premium perché il compito dell'amministratore è incentrato piuttosto alla gestione (creazione/rimozione) e alla modifica, non tanto all'interrogazione specializzata.

4 Uso del polimorfismo

4.1 Uso del polimorfismo della gerarchia utenti

Oltre ai metodi virtuali citati nella precedente sezione, per quanto riguarda l'uso polimorfico della classe User è stato reso virtuale il distruttore per non creare memory leak. Tale metodo è fondamentale affinché non venga lasciato garbage nella memoria. Alla distruzione di un utente viene invocato il distruttore standard della classe concreta dell'utente specifico.

4.2 Uso del polimorfismo della gerarchia prodotti

In riferimento alla gerarchia prodotti vale lo stesso discorso della gerarchia utenti.

4.3 Uso del polimorfismo nella View

La distruzione dei QWidget viene gestita chiamando QWidget::setAttribute(Qt::WA_DeleteOnClose), poiché Qt stessa si occupa della gestione della memoria.

5 Manuale utente

Una volta eseguito l'accesso verrà visualizzata la MainWindow.

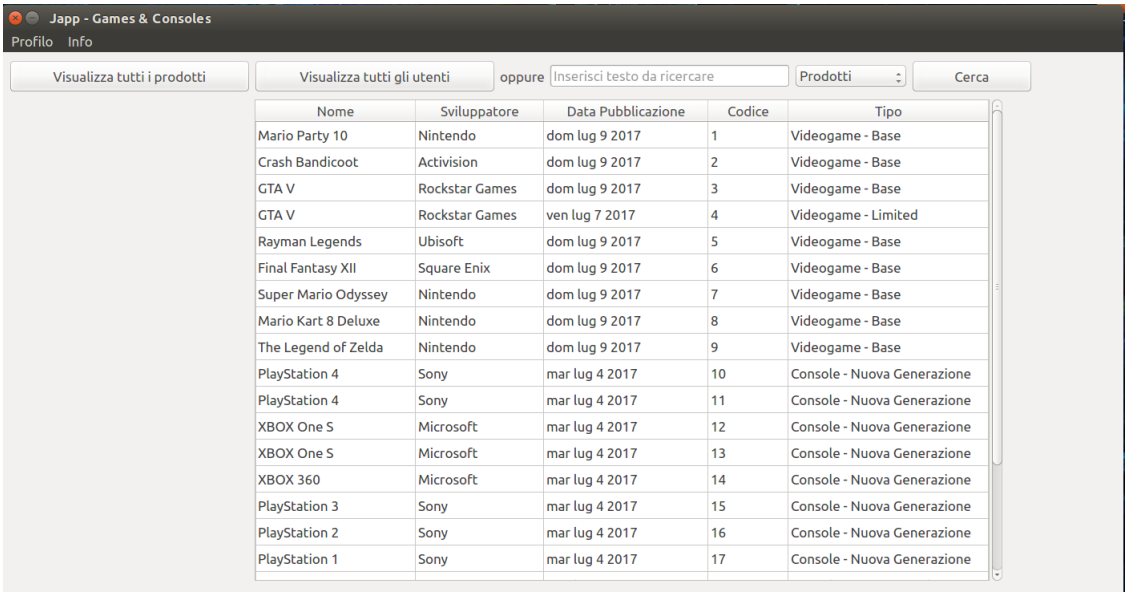

Figura 3: Finestra principale - Accesso con User Base

In tale finestra è disponibile una barra menù con voci **Profilo**, **Aggiungi** e **Info**. Cliccando sulla prima voce si aprirà un menù a tendina con:

- **Impostazioni personali** (in cui sarà possibile modificare le proprie informazioni).
- **Exit**, per effettuare il logout.

La seconda voce è **Aggiungi**, che sarà visibile per l'amministratore con possibilità di aggiunta sia di Prodotti, sia di Utenti; per l'User Premium solo aggiunta Prodotti, e sarà nascosta per l'User Base. Infine, la terza e ultima voce è **Info**, una finestra contenente informazioni sullo sviluppatore (è stata eliminata volontariamente la x di default per la chiusura di finestra in modo tale da chiuderla tramite il tasto OK, evitando la ridondanza della medesima funzione).

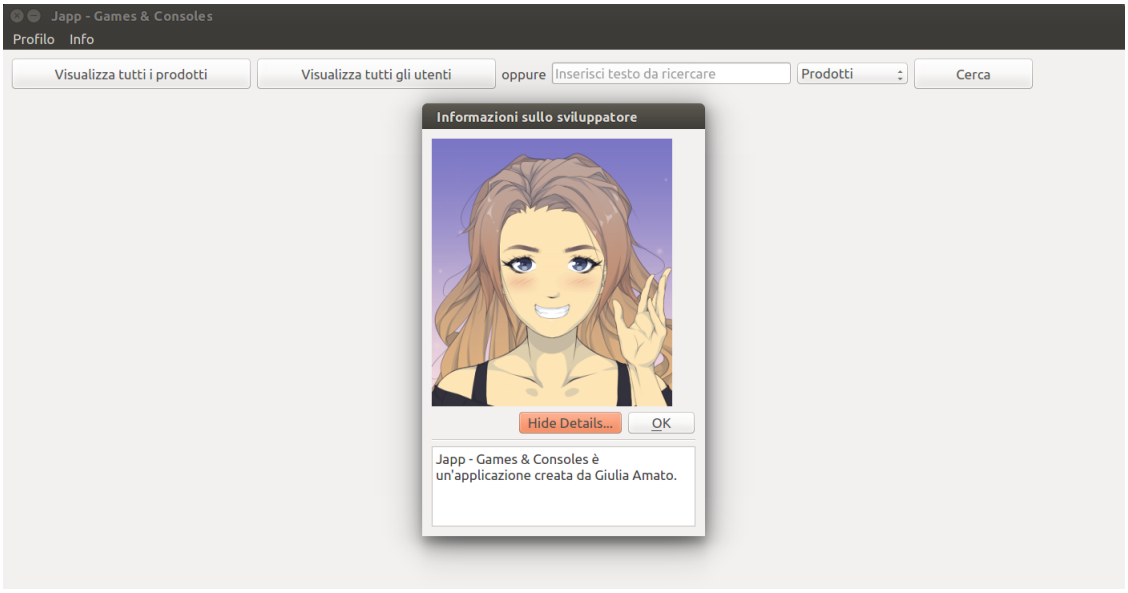

Figura 4: Informazioni sviluppatore

Per arrivare alla consultazione e/o modifica ed eliminazione, se permesse, basterà dunque cliccare su *Visualizza tutti i prodotti* o su *Visualizza tutti gli utenti*, che apriranno le rispettive tabelle con tutte le informazioni salvate, e successivamente fare doppio click sulla riga della tabella interessata. Infine, se si desidera fare una ricerca più specifica basterà utilizzare la barra di ricerca posta sulla sinistra, scrivere una parola (anche parziale) di ciò che si vuole ricercare, selezionare in quale ambito (Utenti o Prodotti) e cliccare su *Cerca*.

5.1 Manuale utente Amministratore

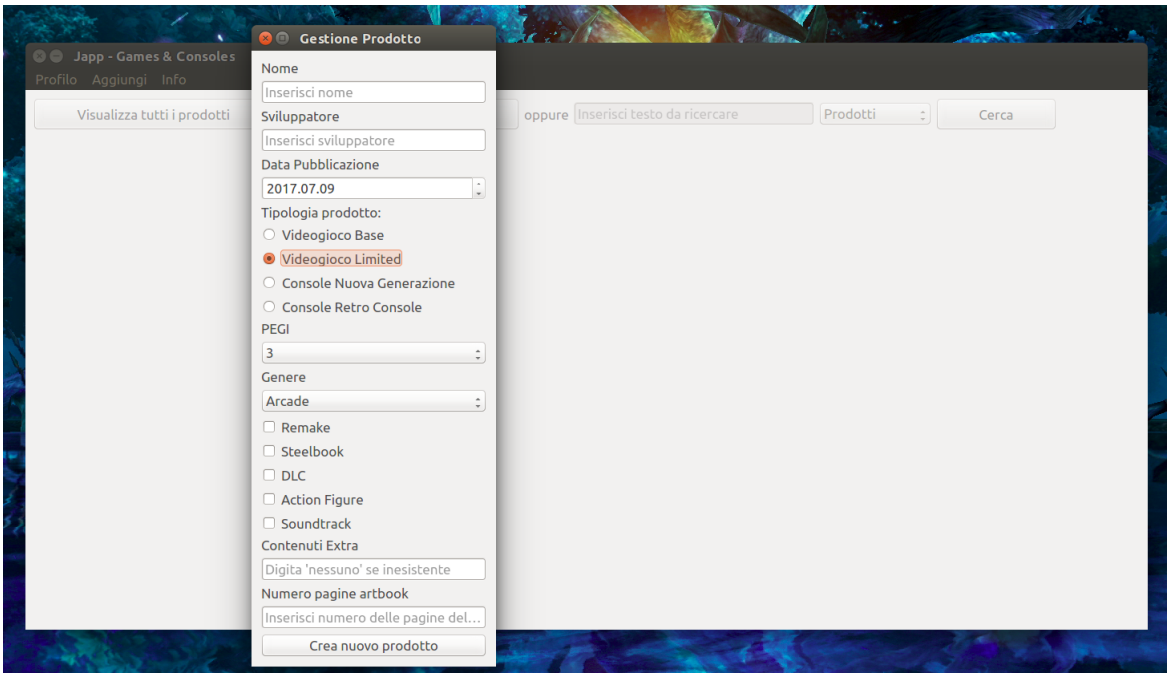

Figura 5: Aggiunta di un prodotto, funzionalità comune Admin e User Premium

Nota: Per aumentare il fattore sicurezza si è deciso che, una volta che l'amministratore ha creato un utente, quest'ultimo può effettuare l'accesso e modificare la propria password. Nel momento in cui l'amministratore dovrà modificare i campi dell'utente nel pannello di modifica, la password verrà oscurata.

5.2 Manuale utente Base e Premium

Se l'accesso viene eseguito da un utente Base allora le funzionalità previste sono:

- `Visualizza tutti i prodotti`, se si effettua il doppio click è possibile consultare tutte le informazioni che vengono aggiunte ai campi comuni dei prodotti.
- `Visualizza tutti gli utenti`, in cui non è ritenuto necessario implementare il doppio click perché tutte le informazioni sono già presenti nella lista
- La possibilità di modifica dei propri dati personali in `Profilo > Impostazioni personali`
- La ricerca semplice tramite il bottone `Cerca`

Se invece viene eseguito l'accesso da un utente Premium otteniamo le stesse funzionalità di un Utente Base (con aggiunta la possibilità di modifica e rimozione, come espresso in precedenza), ma anche l'aggiunta dei prodotti in `Aggiungi > Prodotto` e le possibilità di ricerca addizionali tramite tre QPushButton e una QComboBox, in ordine:

- `Consoles in Ed. Speciale` : Permette di visualizzare nella tabella una lista di prodotti Consoles (New Generation e RetroConsole) che abbiano la caratteristica indicata.

- `Videogames con contenuto extra` : Se nella creazione di un Videogame Limited, alla voce "Contenuti Extra" è stato inserito un testo diverso da "nessuno", come segnalato nel placeholder, visualizza tali prodotti.
- `Videogames versione Remake` : Visualizza tutti i videogames (Base e Limited) che siano un Remake di un titolo precedentemente uscito.
- `Cerca videogames per PEGI` : Per ogni classe del PEGI visualizza tutti i Videogames (Base e Limited).

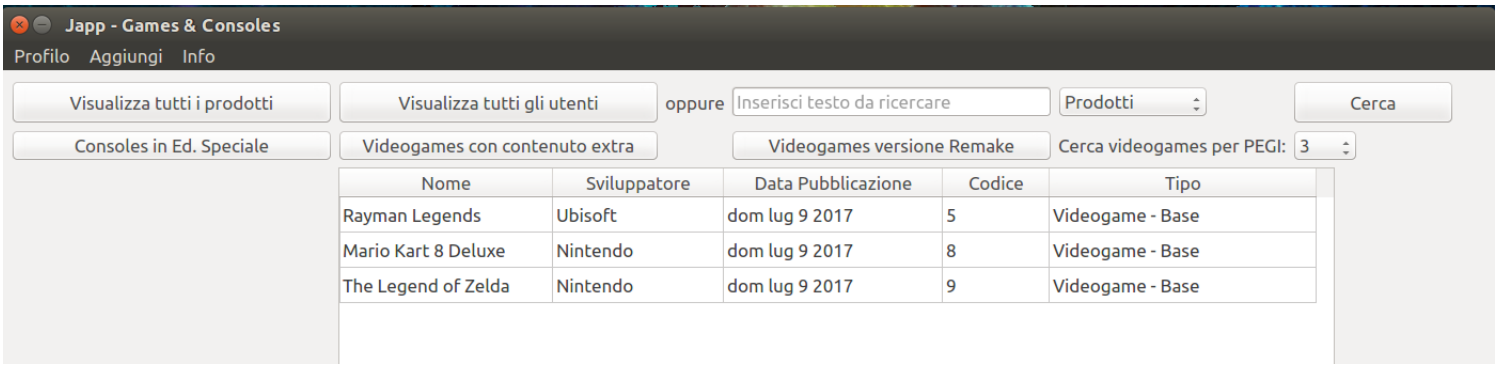

Figura 6: Ricerca riservata all'utente Premium: risultato del QPushButton "Videogames versione Remake"

A Indicazioni conclusive

A.1 Impegno temporale

Progettazione modello e GUI: 3 h
Codifica modello e GUI: circa 50 h
Debugging: 5 h
Testing: 2 h

In modo particolare mi ha richiesto maggiormente tempo lo sviluppo del MVC, sia lato progettuale che lato debugging.

A.2 Informazioni tecniche

Sistema operativo: ubuntu 16.04 LTS
Versione Qt: Qt Creator 3.5.1 based on Qt 5.5.1
Compilatore: GNU g++ 5.3.1