Progetto di programmazione ad oggetti Qontainer

"GameStore"

Nome: Brunetta Alessio

Matricola: 1049202

Anno accademico: 2018/2019

Indice

1 Relazione

1.1 Introduzione

Il progetto è un gestionale per l'organizzazione di videogiochi e carte collezionabili. Attraverso un'interfaccia grafica è possibile gestire gli oggetti memorizzati con operazioni di visualizzazione, modifica, cerca, inserisci e rimuovi. Il progetto è stato sviluppato utilizzando l'ide QtCreator nella versione 5.5.1 su un sistema operativo Windows10. Alla conclusione delle sviluppo è stato testato sulla macchina virtuale della quale è stato fornita l'immagine durante il corso di programmazione ad oggetti.

1.2 Modalità di consegna

Il progetto è stato consegnato attraverso il comando "consegna progetto-pao-2019". All'interno sono presenti tutti i file necessari e il file.pro che si genera automaticamente da Qt all'avvio del progetto. Il file.pro è necessario per la compilazione attraverso "qmake" perché la compilazione attraverso il comando "qmake-project" genera un file non adatto. Questo per permettere la generazione automatica tramite "qmake" del "Makefile".

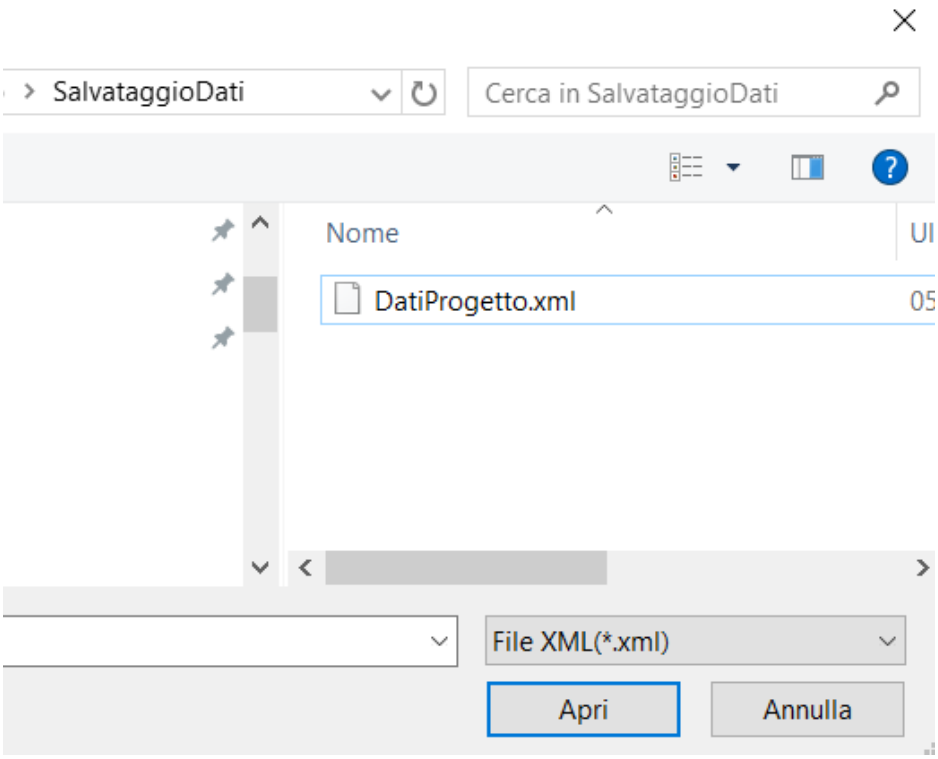

Figura 1: Pagina iniziale del negozio

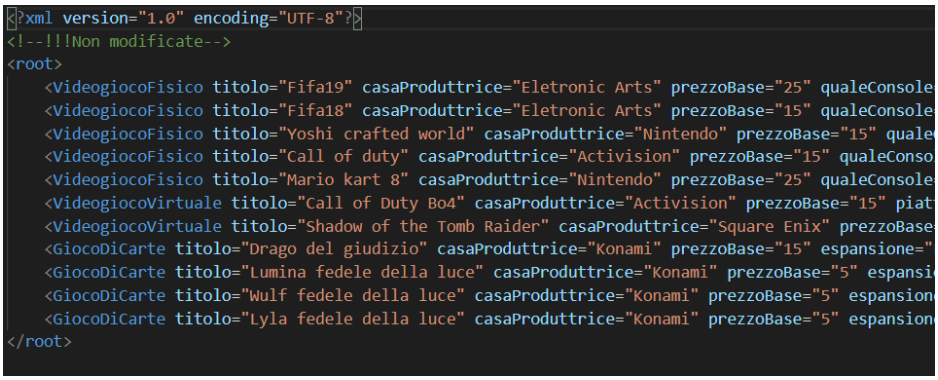

Figura 2: Esempio pagina di xml

1.3 Descrizione classi e gerarchia

La gerarchia del progetto è costituita da una classe base astratta chiamata **itemBase**, essa contiene tutti i campi che in comune per ogni classe derivata. Ci sono tre classi instanziabili che derivano dalla classe base **itemBase** e sono:

- **virtualGame**: gioco non disponibile in copia fisica, ma solo attivazione su determinate piattaforme;
- **physicalGame**: gioco con copia fisica esempio Fifa18 per ps4 con scatola e dvd;
- **cardGame**: tutte le carte collezionabili, compresa carta singola o un inseme di carte esempio una singola carte di Magic e un mazzo già formato.

In seguito la descrizione del contenuto delle classi.

1.3.1 Classe itemBase

- **titolo**: titolo del gioco o della carta o del mazzo;
- **casaProduttrice**: casa che ha prodotto il gioco o le carta/e;
- **prezzoBase**: il prezzo standard e base del gioco, il quale andrà a cambiare nel risultato finale in base alle specifiche di ogni prodotto.

1.3.2 Classe virtualGame

- **piattaForma**: il nome della piattaforma dove il gioco è disponibile es Steam, Origin, Battle;
- **seasonPass**: se il gioco contiene anche il pass stagionale per avere inclusi i futuri aggiornamenti.

1.3.3 Classe physicalGame

- **qualeConsole**: il nome della console dove il gioco è disponibile es Ps4, Ps3, Switch;
- **edizione**: il tipo di edizione del gioco es Standard, Deluxe, Premium.

1.3.4 Classe cardGame

- **espansione**: il nome dell'espansione della carta/e;
- **primaEdizione**: se la carta/e è in prima edizione;
- **numeroCarte**: il numero di carte es 1 o 40 se è uno starter deck;
- **starterDeck**: se la carte fa parte di uno starter deck o se le carte sono uno starter deck.

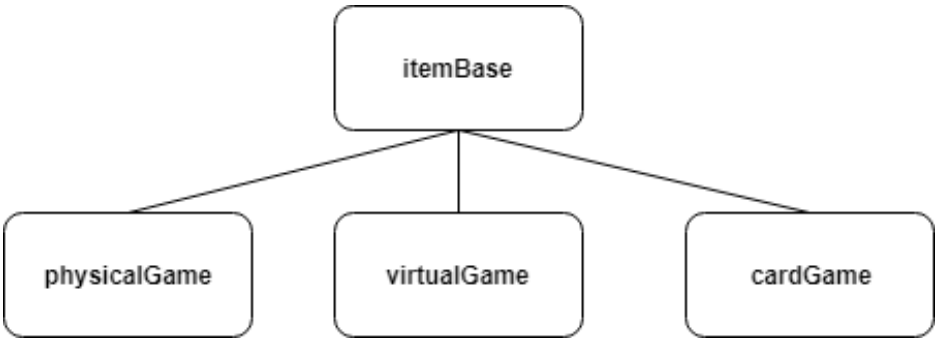

Figura 3: Gerarchia

1.4 Qontainer o contenitore

Per gestire l'inserimento dei dati è stato implementato un template di classe con la funzione di fare da contenitore. Il contenitore essendo templetizzato può accettare qualsiasi tipo di variabile da contenere. Il contenitore è stato implementato come una lista di nodi doppiamente linkata dove ogni nodo contiene tre campi e sono un campo next che contiene il puntatore al nodo successivo, un campo prev che contiene il puntatore al nodo precedente e un campo info templetizzato che contiene l'informazione. Nella classe Contenitore sono presenti due variabili nodo di nome first e last la quali contengono il puntatore al primo e ultimo nodo della lista. Questo per gestire in modo più efficace operazione di inserimento. Le funzioni principali all'interno del contenitore sono la PushBegin e la PushEnd servono per inserire all'inizio o fine della lista. Sono state create due classi atte allo scorrimento della lista, le quali sono Iterator e Constiterator. La classe contenitore inoltre contiene altri funzioni come begin, end, erase le quali sono atte al funzionamento delle classi Iterator e Constiterator. Nello specifico la funzione erase andrà a eliminare e sistemare i nodi della lista in un'operazione di eliminazione.

1.4.1 Model view controller

Per lo sviluppo dell'applicazione è stato usato lo standar model-view-controller.

- **model**: il model fornisce i dati all'applicazione;
- **view**: fornisce la parte grafica dell'applicazione
- **controller**: gestisce i comandi utente e si occupa di collegare i dati del model con la view.

1.4.2 Modello.cpp e Modello.h

Nel modello sono presenti le funzioni principali per il caricamento e salvataggio dei dati da xml.

1.4.3 Controller.cpp e Controller.h

Nel controller viene gestito tutto il funzionamento principale dell'applicazione. Esso va a collegare il modello, cioè dove sono memorizzate le informazioni, con la view dove le va a visualizzare e interfacciare con l'utente.

1.5 Codice estensibile

Il progetto è diviso tra parte grafica e parte gerarchica. Ogni file è diviso in .h e .cpp, la dichiarazioni delle variabili, classi e funzioni è fatta nel proprio file.h mentre il corpo viene implementato nel file.cpp. La maggior parte del codice è stata commentata per chiarire il funzionamento così da rendere migliore un futuro riutilizzo del codice. Nella sviluppo della gerarchia l'eredità implementata renderà un futuro ampliamento di essa facile da fare in quanto basterà aggiungere un nuova classe che deriva dal itemBase. La grafica è stata divisa in più parti, come la pagina negozio, ricerca, modifica e inserisci, questo per rendere più semplice un futuro lavoro su una parte di essa.

1.6 Metodi polimorfi

Nella gerarchia sono stati implementati i seguenti metodi polimorfi:

- **virtual itemBase()**: il distruttore virtuale;
- **virtual double calcolaPrezzo() const**: il metodo virtuale per calcolare il prezzo finale del prodotto, esso parte dalla classe base per poi scendere nel proprio sottotipo così di restituire un prezzo finale calcolato attraverso specifiche caratteristiche univoche di ogni tipo;
- **virtual string getTipo() const**: metodo utilizzato per restituire il tipo di prodotto;
- **virtual bool operator==(const itemBase&)**: operatore di uguaglianza ridefinito in modo da controllare sia un campo e basta o tutti;
- **virtual bool operator!=(const itemBase&)**: operatore diverso ridefinito;
- **virtual string infoItem() const**: operatore usato per restituire la stringa completa di tutte le caratteristiche di ogni oggetto per poi essere inserita nella lista di visualizzazione;
- **std::ostream& operator«(std::ostream& , const itemBase&)**: operatore di stampa ridefinito, utilizzato solo nell'uso da terminale dell'applicativo.

1.7 Avvio del progetto

All'avvio dell'applicazione il programma chiede di selezionare un file in formato .xml, questo file contiene tutti i dati che saranno caricati all'interno del nostro contenitore. Inoltre su que-

sto file si salveranno i dati relativi a nuovi inserimenti, eliminazioni e modifiche dei prodotti. Questo è possibile attraverso l'uso del file xml. Xml è un linguaggio di markup che consente la rappresentazione di documenti e dati strutturati su supporto digitale.

1.8 Interfaccia grafica

L'interfaccia grafica è composta da tre tab principali e una nuova pagina che si apre all'occorrenza di una modifica e sono:

- **Negozio**;
- **Ricerca**;
- **Inserisci**;
- **Pagina di modifica**.

1.8.1 Negozio

Questa è la pagina principale dell'applicazione, qui è possibile visualizzare una lista di tutti i nostri prodotti memorizzati, si possono visualizzare tutti assieme o divisi per categoria. Ogni oggetto è selezionabile, dopo aver premuto su un item nella lista sarò possibile premere uno dei due pulsanti di elimina e modifica. Il pulsante elimina procederà all'eliminazione dell'oggetto con annessa conferma di avvenuta eliminazione o in caso negativo avvertirà che non è andato a buon fine. Il pulsante modifica aprirà una pagina atta alla modifica dell'oggetto selezionato.

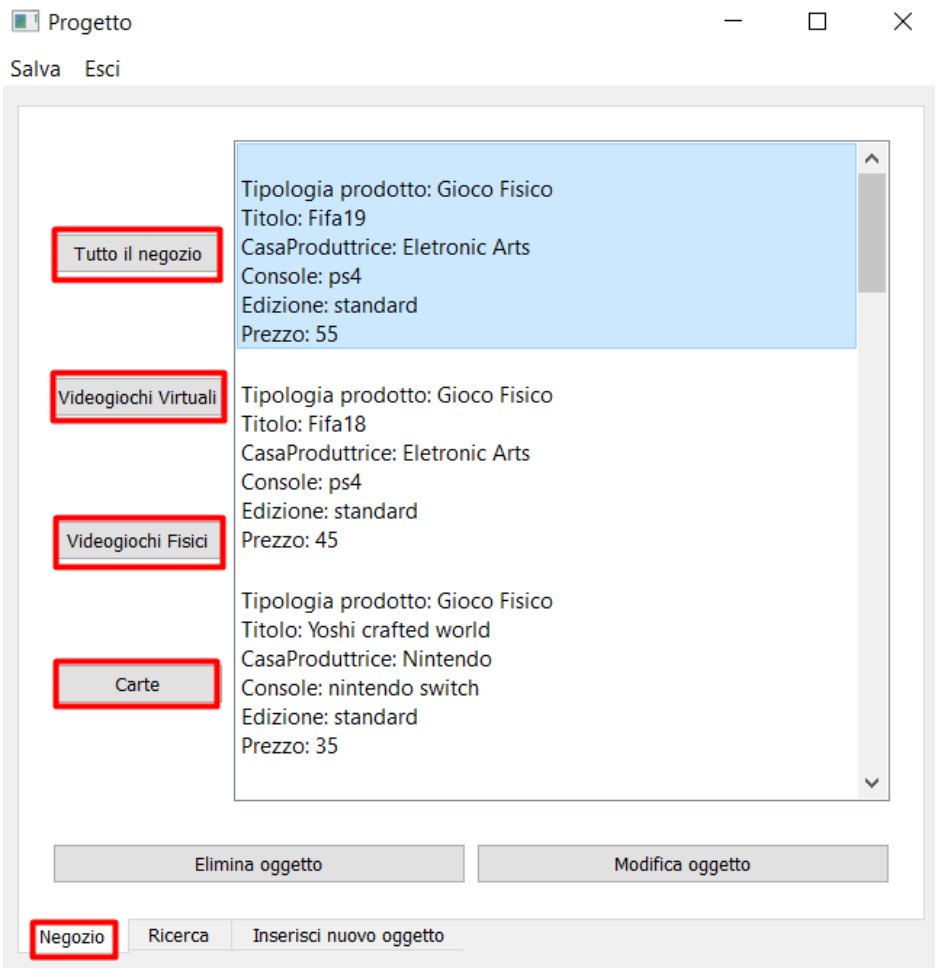

Figura 4: Pagina principale del negozio

1.8.2 Ricerca

Attraverso questa pagina è possibile ricercare un oggetto memorizzato nel nostro contenitore. Questa pagina à dinamica e cambierà in base al radiobutton selezionato. Si può effettuare la ricerca con un solo parametro o più inseriti. Per la ricerca di un item viene sfruttato l'overloading dell'operatore di uguaglianza modificato in modo da poter ricercare uno o più campi in base a quelli inseriti. La ricerca può trovare da zero a più elementi corrispondenti ai parametri inseriti, in caso positivo sarà comunicato l'esito positivo mentre in caso di nessun match sarò comunicato l'esito negativo.

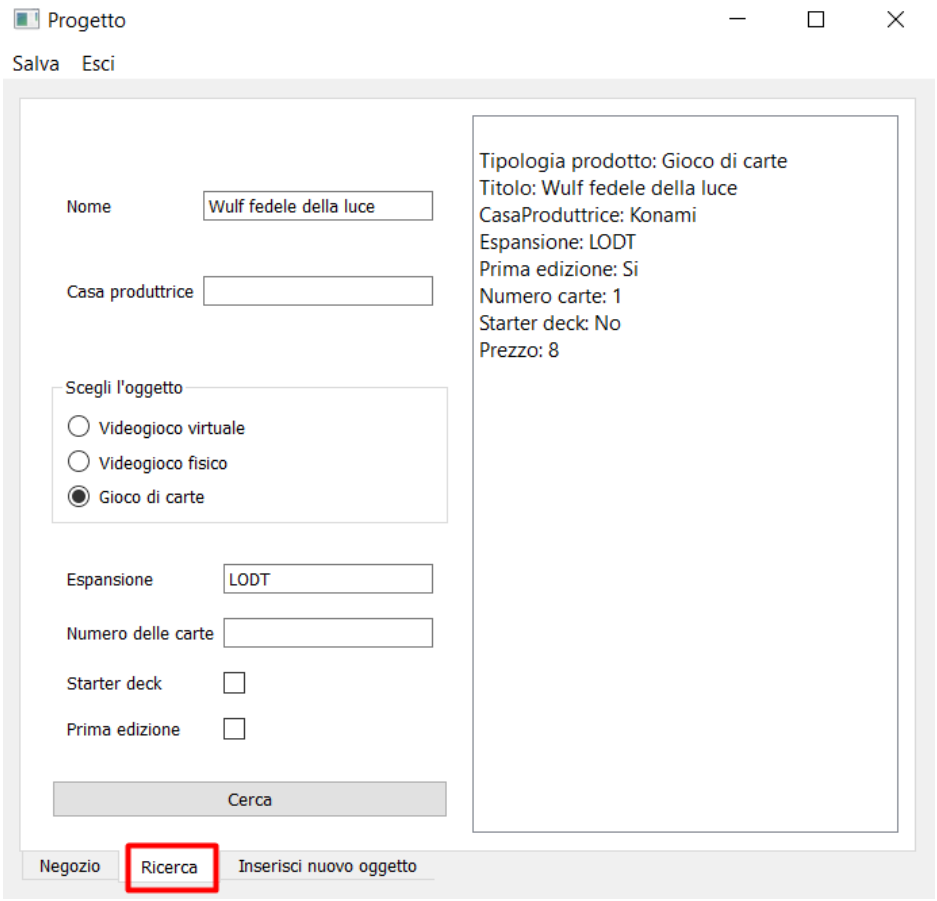

Figura 5: Pagina di ricerca

1.8.3 Inserisci

Attraverso questa pagina è possibile inserire un nuovo oggetto nel contenitore. Come la pagina precedentemente descritta anche questa risulta essere dinamica, cambia le specifiche di inserimento del sottotipo in base al radiobutton selezionato. Per un nuovo inserimento tutti i campi devono essere compilati e in modo corretto. Se l'inserimento viene effettuato sarà comunicato l'esito positivo, in caso non risulti essere inserito sarà comunicato l'esito negativo.

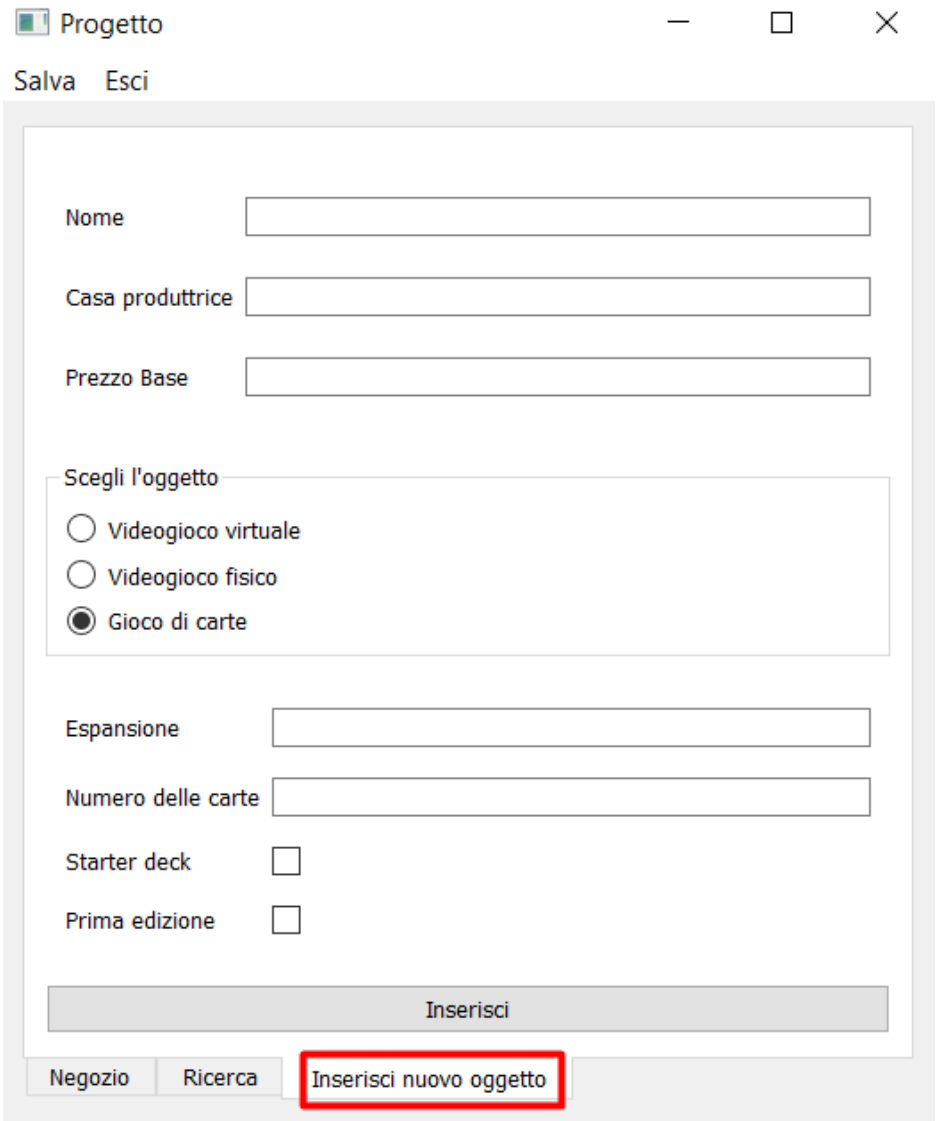

Figura 6: Pagina di inserimento

1.8.4 Pagina di modifica

Questa pagina compare alla pressione del tasto modifica con un elemento selezionato. Si aprirà con tutti campi già compilati in modo da semplificare la modifica di essi. Per una modifica tutti i campi devono essere compilati e in modo corretto. Se l'inserimento viene effettuato sarà comunicato l'esito positivo, in caso non risulti essere inserito sarà comunicato l'esito negativo. Quando questa pagina è aperta non sarà possibile interagire con il resto dell'applicazione, alla chiusura il funzionamento riprenderà.

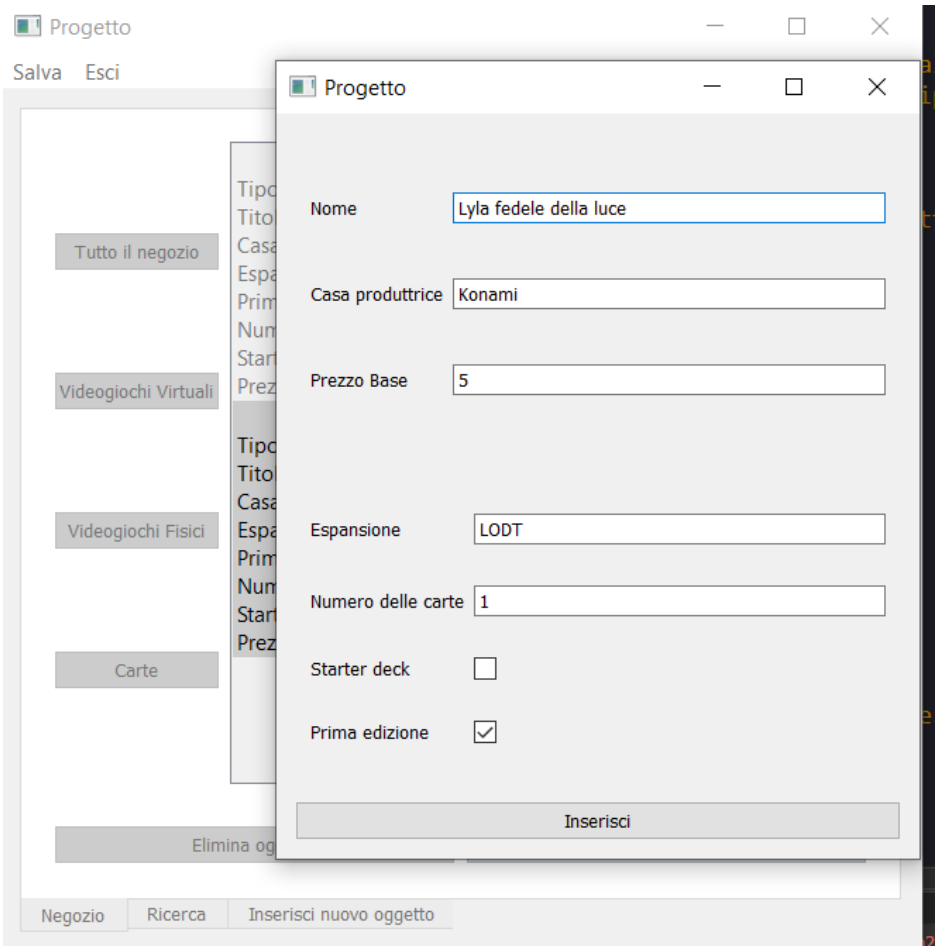

Figura 7: Pagina di modifica

1.9 Ore di lavoro

Periodo	Ore
Analisi dei requisiti	1
Pianificazione contenitore e gerarchia	3
Implementazione contenitore e gerarchia	8
Apprendimento gestione xml	2
Implementazione e apprendimento Gui	35
Debug su macchina virtuale	4
Scrittura relazione	3
Totale	56