Relazione per il progetto di Programmazione ad oggetti

(A.A 2018/2019)

Grigoras Valentin

Matricola: 1099561

Contenuti

Elenco delle figure

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto QONTAINER è lo sviluppo in C++/Qt di un gestore di un contenitore di oggetti di una gerarchia di tipi tramite una interfaccia utente grafica (GUI).

1.2 Template di classe Container <T >

Il contenitore utilizzato è una *lista doppiamente linkata* composta da un insieme di elementi detti **nodi**, collegati linearmente tra di loro. Ogni nodo è un piccolo contenitore di dati. All'interno della lista doppiamente linkata, ogni nodo contiene un riferimento al nodo precedente (**prev**) e a quello successivo (**next**), mentre la lista terrà solo un riferimento al primo e all'ultimo nodo (chiamati rispettivamente **first** e **last**). Si è scelto di utilizzare una lista concatenata per i suoi vantaggi principali, ossia cancellazione e inserimento in tempo costante

1.3 Descrizione dell'applicazione

L'applicazione consiste in un contenitore di oggetti capace di gestire dei videogiochi. L'applicazione consente di creare e gestire nuovi oggetti e di salvarli/caricarli su/da un file XML.

2 Compilazione ed esecuzione

Il progetto prevede l'utilizzo di un file P2Qantainer.pro diverso da quello ottenibile con l'esecuzione qmake -project, a causa della presenza di funzionalità di c++11, come le keywords default ed override. Inoltre è stato aggiunto un file default.xml che può (eventualmente) essere aperto dall'applicazione e che contiene alcuni dati di test pronti all'uso. L'intero progetto è stato realizzato su sistema operativo Windows 10 Enterprise e Qt creator 5.9 con compilatore MinGW 5.3.0.

3 Descrizione aspetti progettuali

3.1 Model View Controller

L'applicazione è stata sviluppata secondo il design pattern Model View Controller. Il MVC è un sistema di progettazione dove la struttura dei programmi che manipolano dati è scomposta in tre livelli:

- **Model**: rappresenta il modello dei dati;
- **View**: rappresenta la visualizzazione dei dati quindi l'interfaccia utente;
- **Controller**: rappresenta il controllo delle risposte alle azioni dell'utente.

4 Gerarchia dei tipi

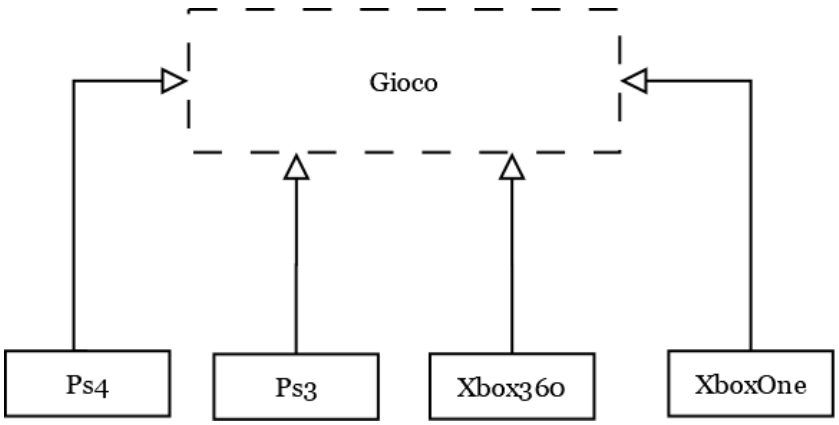

Figura 1: Gerarchia dei tipi

La classe base astratta polimorfa Gioco rappresenta la classe base della gerarchia e rappresenta qualsiasi oggetto astratto che può essere inserito nel contenitore. La classe gioco è astratta perchè possiede il seguente metodo virtuale puro:

```
public:
   virtual string getTipo()const =0;
```

Il metodo getTipo() ritorna il tipo del gioco in formato stringa. Di seguito viene mostrato un esempio di implementazione per la classe Ps3:

```
std::string Ps3::getTipo() const
{
     return "Ps3";
}
```

La classe Gioco fornisce inoltre un metodo d'istanza utile per il popolamente della GiochiListItem, una delle classi delal view che verrà trattata successivamente. Il metodo getInfo() ritorna una stringa contenente la concatenazione del contenuto di tutti i campi dati. Questo metodo fornisce all'utente una visione ampia del contenuto di tutti i giochi presenti nel contenitore.

```
/*prototipo*/
string getInfo()const;

/*implementazione*/
std::string Gioco::getInfo() const
{
    stringstream stream;
    string print_multiplayer =
         std::string(getMultiplayer() ? "Si" : "No");
    string print_online = std::string(getOnline() ? "Si" : "No");
    string print_s4k = std::string(get4k() ? "Si" : "No");
    string str= "";
    return str.append("\nTipo: " +getTipo())
```

```
                .append("\nNome: " +getNome())
                .append("\nAnno di rilascio: "+ getAnnoRilascio())
                .append("\nSviluppatore: " + getSviluppatore())
                .append("\nMultiplayer: " +  print_multiplayer)
                .append("\nOnline: " + print_online)
                .append("\n4k: "+  print_s4k)
                .append("\nPegi: " + getClassificazionePegi())
                .append("\nGenere: " + getGenere())
                .append("\nDescrizione: " + getDescrizione());
}
```

La gerarchia è stata pensata per descrivere al meglio la realtà. Infatti basti pensare che i giochi hanno le stesse caratteristiche a meno della piattaforma a cui sono destinati. Il metodo virtuale puro serve a ritornare la "piattaforma" giusta associata ad ogni tipo di gioco. La classe Gioco contiene tutti i campi dati utili alla descrizione e costruzione di un generico gioco. Il loro scopo è il seguente:

- string **nome**: rappresenta il nome del gioco;
- string **annoRilascio**: rappresenta l'anno in cui il gioco è uscito;
- string **genere**: rappresenta il tipo del gioco selezionabile da una combo box;
- string **classificazionePegi**: PEGI è l'acronimo di Pan European Game Information e rappresenta la classificazione del videogioco in base all'età. Questo valore è selezionabile da una combo box;
- string **sviluppatore**: rappresenta l'ente o l'azienda che ha sviluppato il gioco;
- bool **multiplayer**: rappresenta l'esistenza della modalità multiplayer non online;
- bool **online**:rappresenta l'esistenza della modalità multiplayer online;
- bool **supporto_4k**: rappresenta la possibilità di poter giocare il gioco con una risoluzione 4k;
- string **descrizione**: rappresenta la descrizione del gioco.

Una possibile estensione con un'ulteriore livello di astrattezza è la seguente:

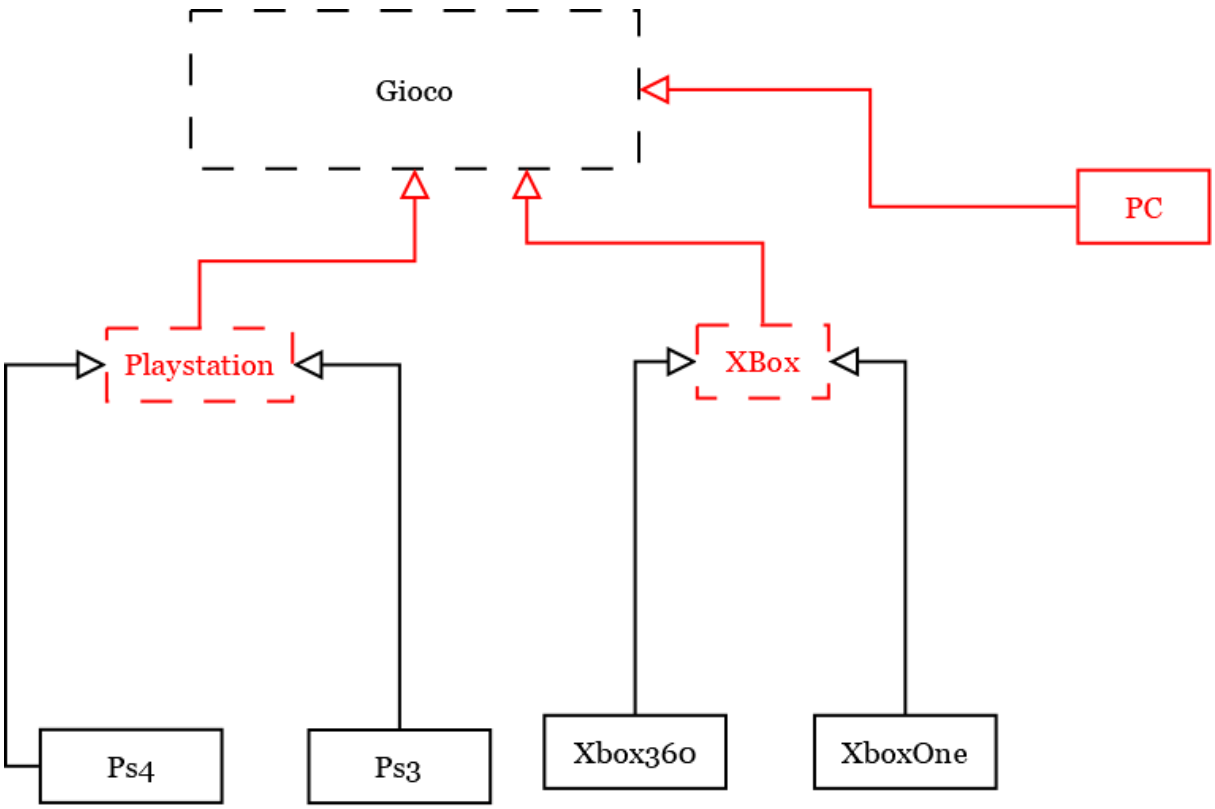

Figura 2: Estensione della gerarchia

4.1 Uso del polimorfismo

Il polimorfismo è stato applicato nella gerarchia con il metodo virtuale puro GetTipo() per ritornare le informazioni giuste per ogni tipo di gioco, e attraverso il distruttore virtuale avente comportamento di default.

5 GUI

L'applicazione è composta da un'interfaccia di visualizzazione e crezione giochi e un'altra che permette la semplice ricerca attraverso dei filtri. Queste interfacce ereditano da QWidget e vengono descritte di seguito:

- **LayoutInserisci:** questa interfaccia è composta da tutti i campi necessari alla creazione di un gioco. Poichè alcuni campi vengono utilizzati anche per la ricerca ho preferito creare le mie quattro classi combo box che ereditano da QComboBox: *ComboBoxAnno*, *ComboBoxGenere*, *ComboBoxPegi*, e *ComboBoxTipo*. In questo modo, in caso di necessità future, posso estendere e riutilizzare le classi appena citate.
- **LayoutVisualizza**: questa interfaccia visualizza i giochi presenti nel contenitore. L'utente può modificare ed eliminare gli elementi presenti all'interno del contenitore.
- **LayoutRicerca**: questa interfaccia permette il filtraggio degli elementi del contenitore. La ricerca avviene tramite la creazione di una lista d'appoggio in modo da tenere separati gli elementi ritornati dalla ricerca dagli elementi presenti nel contenitore.

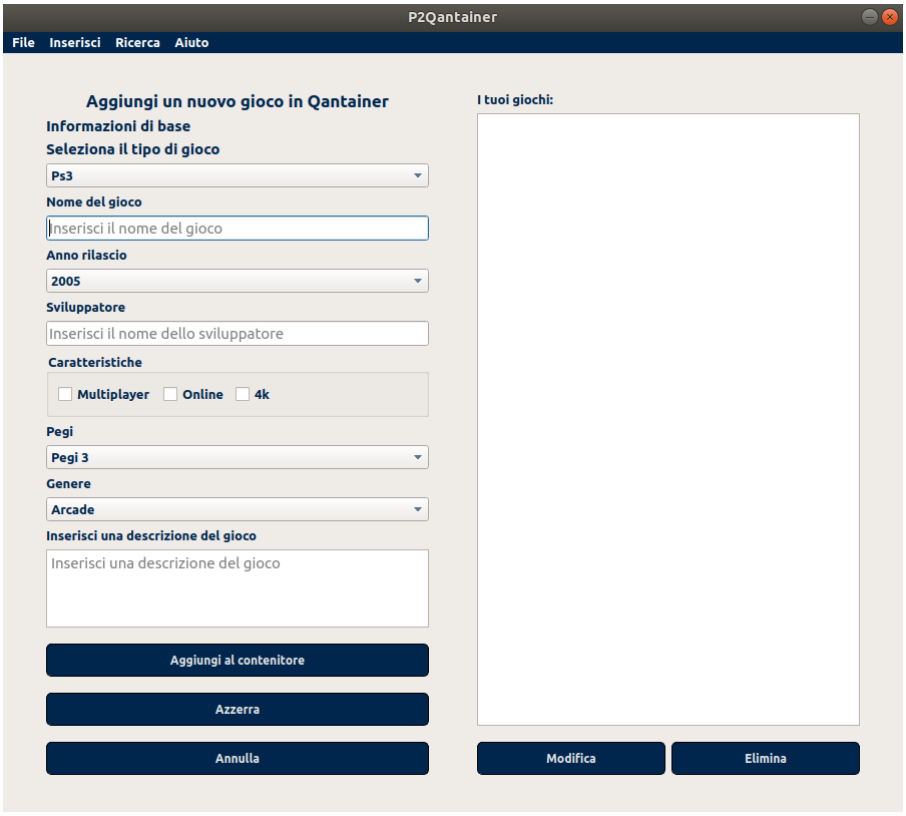

Figura 3: Da sinistra a destra: LayoutInserisci e LayoutVisualizza

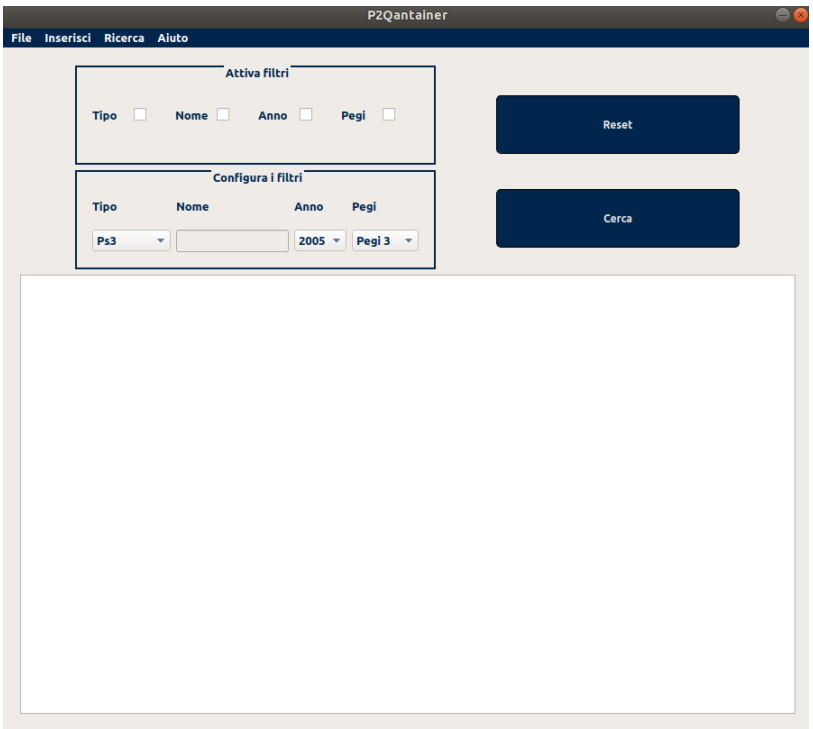

Figura 4: LayoutRicerca

6 Tempo impiegato

Sono state impiegate circa 50 ore, di cui:

- Analisi del problema: 2h
- Progettazione modello: 3h
- Progettazione GUI: 3h
- Apprendimento libreria QT: 6h
- Codifica modello: 12h
- Codifica GUI: 10h
- Debugging: 2h
- Testing: 2h
- Tutorato: 10h