Università degli Studi di Padova


Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica


# CrownSouls

## CrownSouls

Progetto di Programmazione ad Oggetti


Enrico Buratto
1142644


---

Anno Accademico 2019-2020

# Indice

# 1 Introduzione

## 1.1 Abstract

Si vuole realizzare un programma per la gestione di un inventario giocatore per il gioco *Action RPG* "Dark Souls". Il giocatore possiede svariati elementi di diversi tipi; questi elementi possono infatti essere:

**Armature**: oggetti indossati dal giocatore per aumentare la resistenza al danno inflitto dai nemici;

**Armi**: oggetti utilizzati dal giocatore per infliggere danno ai nemici;

**Anelli**: oggetti utili al giocatore per l'aumento delle proprie statistiche; una volta indossati nel gioco, il giocatore vedrà aumentate alcune statistiche personali o delle armi;

**Scudi**: oggetti utilizzati dal giocatore per ridurre il danno dai colpi nemici;

**Guanti**: oggetti utilizzati sia come armatura, poiché aumentano la resistenza al danno, sia come arma, poiché permettono di infliggere danno;

**Scudi d'attacco**: oggetti utilizzati sia come scudo, poiché riducono il danno dai colpi nemici, sia come arma, poiché permettono di infliggere danno.

Un inventario è composto da un insieme di oggetti appartenenti alle diverse tipologie; ogni oggetto presente nell'inventario possiede delle caratteristiche tecniche proprie della categoria di appartenenza.
Il programma deve poter simulare un inventario di questo tipo, permettendo l'inserimento, la rimozione e la visualizzazione degli oggetti e delle loro proprietà.

## 1.2 Funzionalità

Per facilitare la visualizzazione degli oggetti dell'inventario, questi sono suddivisi all'interno del programma in quattro diverse schede; ogni scheda rappresenta una sottosezione dell'inventario, e mostra al suo interno solo gli elementi appartenenti alla categoria indicata dal titolo. Per la gestione degli oggetti dell'inventario sono presenti le seguenti funzionalità:

- Caricamento ed esportazione dell'intero inventario da e su file XML;
- Aggiunta di un nuovo oggetto all'inventario;
- Modifica di un elemento già presente nell'inventario;
- Rimozione di un elemento dell'inventario;
- Rimozione di tutti gli elementi presenti;
- Visualizzazione di oggetti dell'inventario divisi per categoria di appartenenza;
- Visualizzazione delle caratteristiche di ogni elemento dell'inventario, comprese alcune statistiche calcolate automaticamente dal programma;
- Visualizzazione di avvisi d'errore.

## 2 Progettazione

Lo sviluppo del progetto si è basato sul pattern **Model-View** di *Qt* e metodologia mista *top-down* e *bottom-up*.
Oltre alla gerarchia, è stato realizzato un Container templatizzato per il contenimento degli oggetti appartenenti alla gerarchia. Sono stati realizzati inoltre:

- Una GUI (Graphical User Interface), basata su classi preesistenti di Qt;
- Un Model, il quale si occupa della gestione dei dati del programma, basato anch'esso su classi preesistenti di Qt;
- Un filter proxy, che funge da intermediario tra model e view e permette di filtrare i dati per la visualizzazione corretta su ogni tab;
- Una classe di Input/Output su file XML.

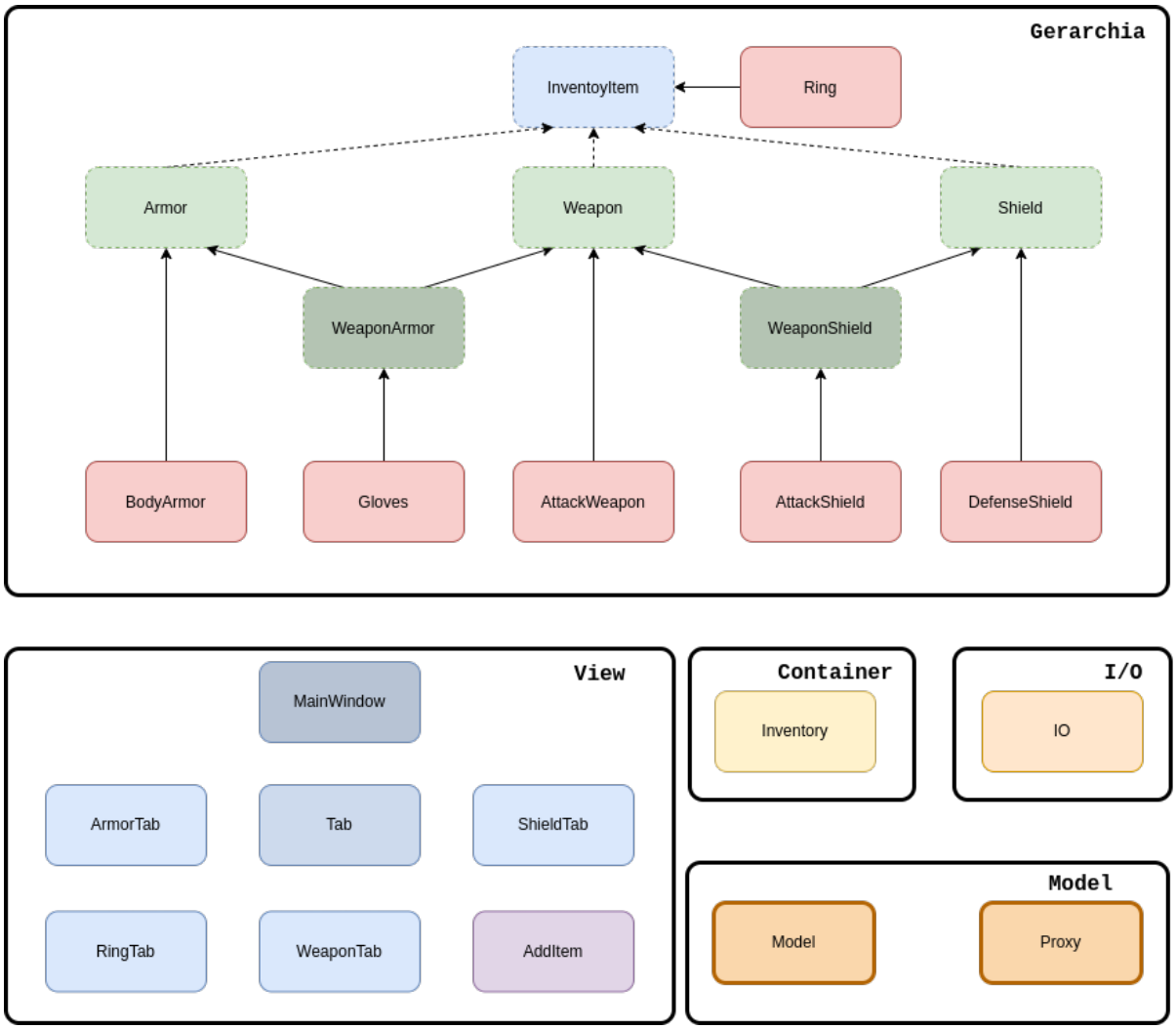


Figura 1: Diagramma delle classi di CrownSouls.

## 2.1 Gerarchia

La gerarchia è composta dalla classe base astratta *InventoryItem*, dalla quale deriva **direttamente** la classe concreta *Ring* e **virtualmente** le classi astratte *Armor*, *Weapon* e *Shield*. Da queste tre classi derivano **singolarmente** e **direttamente** le tre rispettive classi concrete *BodyArmor*, *AttackWeapon* e *DefenseShield*. Viene inoltre utilizzata l'*ereditarietà multipla* per la definizione di classi che rappresentano oggetti appartenenti a più tipi; nello specifico, la classe *WeaponArmor* deriva direttamente da *Armor* e *Weapon*, e la classe *WeaponShield* deriva direttamente da *Weapon* e *Shield*. Questa forma di ereditarietà multipla è di tipo **is-a**, poiché un oggetto WeaponArmor è sia un oggetto Weapon che un oggetto Armor (e lo stesso vale per WeaponShield). Da queste due classi astratte derivano poi rispettivamente le classi concrete *Gloves* e *AttackShield*.
Ciascuna classe implementa dei metodi virtuali che riguardano l'impostazione e il recupero delle diverse proprietà degli elementi, e dei metodi virtuali specifici di ogni sottoclasse astratta per il calcolo e l'ottenimento di statistiche basate sulle proprietà. L'utilizzo del polimorfismo in tale contesto viene illustrato in seguito.

## 2.2 Container

È stata implementata una classe *Inventory* che funge da container. La classe fornisce un template di smart pointer, e simula una lista singolarmente linkata con alcuni accorgimenti: a differenza di una lista singolarmente linkata standard, infatti, essa fornisce anche un puntatore all'ultimo elemento, e permette l'accesso diretto in sola lettura a un dato elemento tramite l'overloading dell'operatore accesso agli elementi del puntatore ([]).
La classe *Inventory* contiene al suo interno due classi annidate:

- La classe *SmartP* rappresenta uno smart pointer; è infatti questa classe a rappresentare un elemento del container di tipo T templatizzato. Oltre al contenuto effettivo dell'elemento e al puntatore all'elemento successivo, la classe è fornita anche di:
  - Costruttore e costruttore di copia profondo;
  - Distruttore profondo;
  - Assegnazione profonda;
  - Overloading degli operatori dereferenziazione e accesso a membro;
  - Overloading degli operatori booleani uguaglianza e disuguaglianza.
- La classe *Iterator* rappresenta l'iteratore del container. Nello specifico, un oggetto di classe Iterator è un iteratore costante, poiché non permette il side effect degli oggetti a cui punta. Questa classe presenta l'overloading dei seguenti operatori:
  - Incremento prefisso;
  - Dereferenziazione;
  - Accesso a membro;
  - Uguaglianza e disuguaglianza.

La classe container fornisce diverse funzionalità di inserimento, cancellazione e ricerca; essa infatti permette:

- L'inserimento e la rimozione di oggetti in testa, in coda o in una posizione data;

- La modifica di un oggetto a una posizione data (sovrascrittura);
- La lettura di oggetti in testa, in coda o a in una posizione data grazie all'overloading dell'operatore [].

### 2.3 Modello

Modello

### 2.4 GUI

GUI

### 2.5 I/O

Il programma permette la lettura e la scrittura di interi inventari. Questa possibilità è data dalla classe *IO*, la quale fornisce i metodi necessari all'input e all'output dei dati tramite file .xml. Questi metodi risolvono il problema specifico del programma sviluppato, e non sono pertanto applicabili ad altri problemi.

### 2.6 Polimorfismo

Polimorfismo

### 2.7 Scelte progettuali

- Si è scelto di utilizzare una lista singolarmente linkata per la facilità e l'efficacia di questa struttura dati in un problema come quello in oggetto. Sono stati però seguiti degli accorgimenti per la semplificazione dell'accesso in sola lettura dati (tramite operatore []) e per la diminuzione dello sforzo computazionale. Un esempio di questo è la presenza di un puntatore all'ultimo elemento, che permette la riduzione di alcune operazioni, tra cui l'aggiunta e la rimozione in coda, da tempo O(n) a tempo costante;
- Si è scelto di optare per una classe di Input/Output non scalabile per questioni di tempo. Una classe facilmente adattabile a più problemi, infatti, avrebbe richiesto uno studio più approfondito e un utilizzo pesante di polimorfismo sulla gerarchia; questo è certamente auspicabile, ma purtroppo incompatibile con le tempistiche reali del progetto.

# 3 Gestione di Progetto

## 3.1 Suddivisione del lavoro progettuale

Il progetto è stato iniziato

## 3.2 Timeline individuale

## 3.3 Ambiente di lavoro

Il progetto è stato sviluppato con Qt Creator v4.11.2 e Atom v1.47.0 su sistema operativo Arch Linux; il compilatore usato è stato GCC v10.1.0. Essendo il compilatore e la versione di Qt del sistema di sviluppo più aggiornati rispetto alle specifiche, sono stati effettuati frequenti allineamenti con la macchina virtuale, contenente Ubuntu 18.04 LTS con GCC alla versione v7.4.0 e Qt alla versione v5.9.5, per verificare la compatibilità di quanto prodotto.

## 3.4 Istruzioni di compilazione ed esecuzione

Il progetto prevede la compilazione tramite file .pro e tool qmake. Le istruzioni per la compilazione e l'esecuzione sono quindi le seguenti:

```
\$ qmake CrownSouls.pro
\$ make
\$ ./CrownSouls
```

Viene inoltre fornito un file .xml, locato nella cartella `extra`, contenente un inventario precompilato di prova.