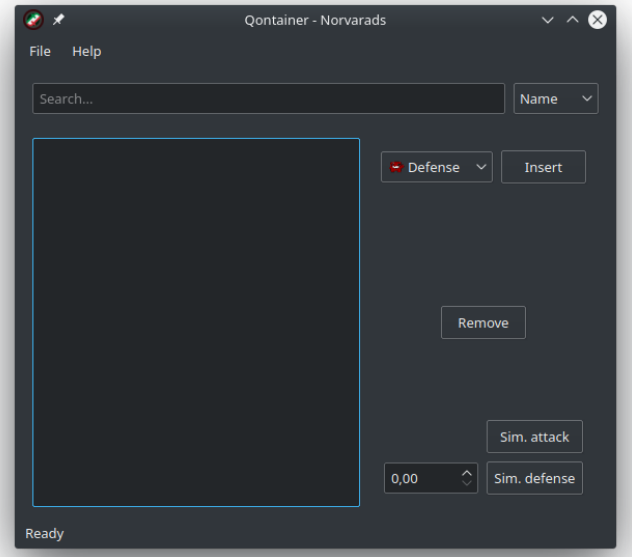
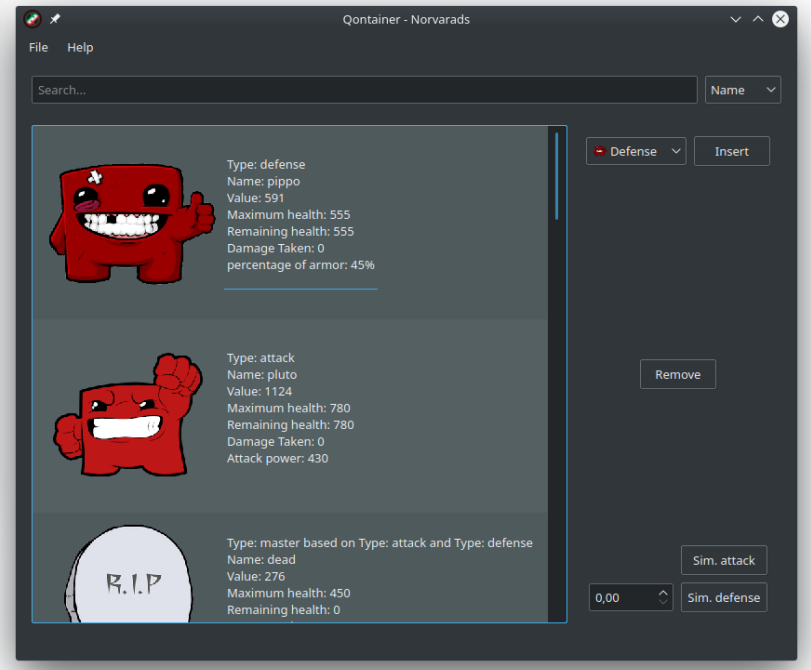
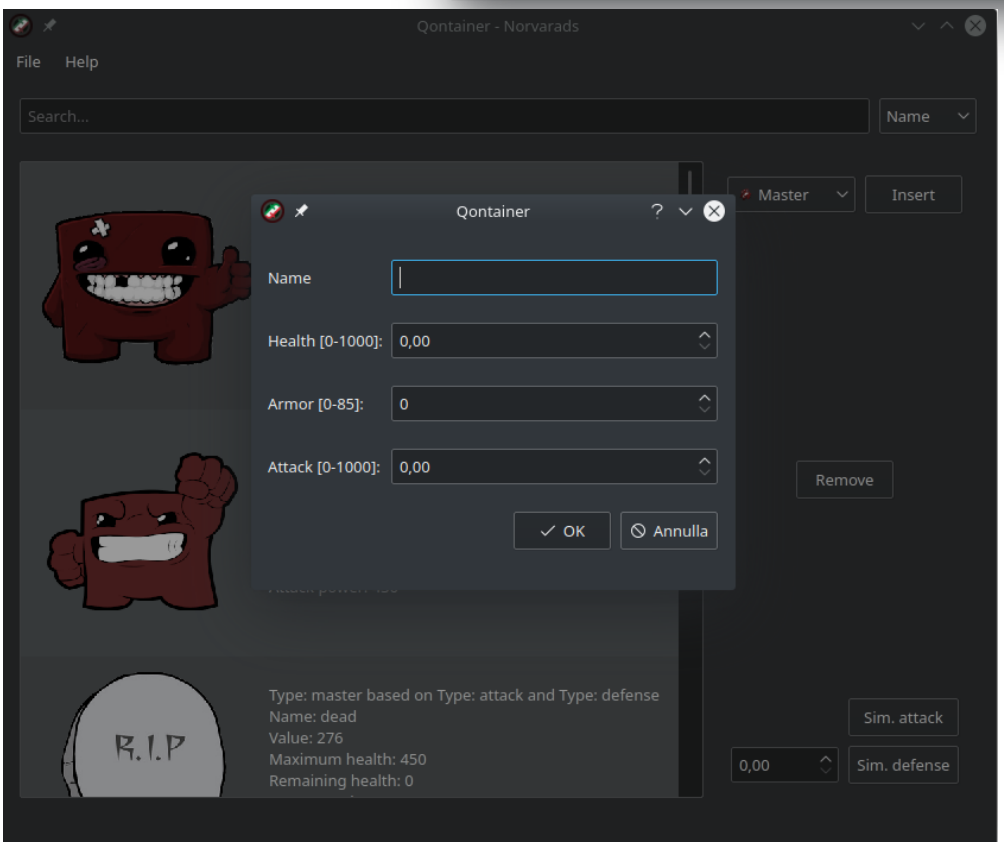
Marco Dalla Libera

Qontainer – Norvarads







Introduzione

Questo progetto realizza un container in grado di archiviare e gestire dei ‘Norvarad’, personaggi di fantasia che fanno parte di un gioco. Ogni Norvarad di base è caratterizzato da un nome ed una salute e possiede poi caratteristiche peculiari per ogni tipo specializzato. Tramite la GUI è possilbile inserire, cercare, rimuovere e modificare i singoli elementi inseriti ed inoltre si possono simulare attachi e difese sui singoli personaggi. Ogni classe di Norvarad ha una sua immagine specifica e nel caso la salute residua diventi zero allora verrà visualizzata una lapide.

È inoltre possibile, tramite il menù ‘File’, salvare l’attuale stato del container su file o caricare uno dei salvataggi eseguiti. In alternativa al menù ‘File’ si possono usare le shortcut ‘Ctrl+O’ e ‘Ctrl+S’.

Compilazione

Per compilare il progetto è neccessario utilizzare il file .pro fornito, in quanto prevede l’uso di funzionalità di C++11 e la configurazione delle directory di inclusione file. I comandi da eseguire saranno quindi solamente qmake e make.

Viene inoltre fornito un file “savedata.xml” contenente dei dati di test da caricare tramite GUI, il file può essere caricato tramite il menu ‘File’ o la shortcut ‘Ctrl+O’ e selezionandolo poi tramite il file chooser.

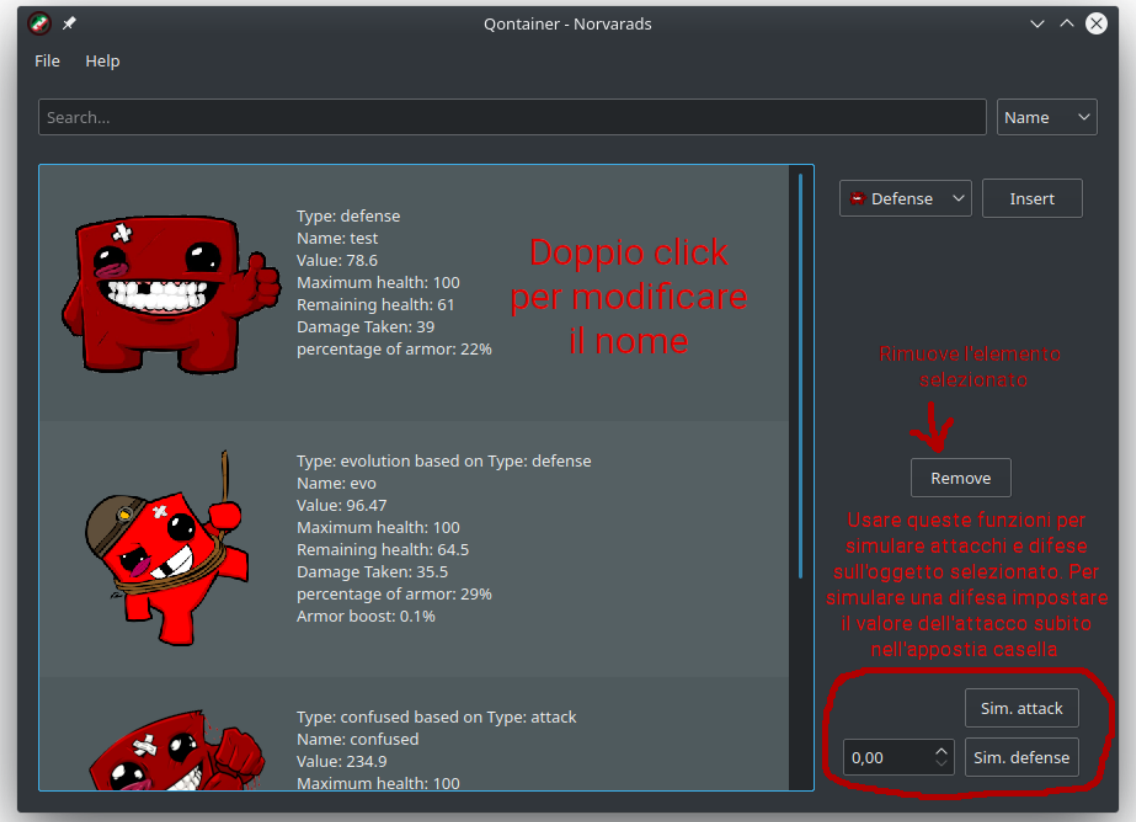
Il progetto è stato realizzato su sistema operativo openSUSE Tumbleweed con Qt 5.12.3, Qt creator 4.8.2 e GCC 8.3.1 20190226.

Tempo impiegato

Circa 52 ore, di cui

- Analisi dei requisiti e progettazione iniziale: 5h
- Progettazione e realizzazione contenitore: 10h
- Progettazione e realizzazione DeepPtr: 3h
- Progettazione e realizzazione gerarchia classi: 8h
- Progettazione modello e GUI (escluso tutorato): 2h
- Analisi e studio della libreria Qt (escluso tutorato): 5h
- Realizzazione di modello e GUI: 15h
- Debug e test (oltre a quelli eseguiti nelle singole fasi): 2h
- Stesura relazione: 2h

Uso GUI



Per le ricerche: selezionare il filtro desiderato e scrivere il valore che si desidera cercare, la lista si aggiornerà dinamicamente. Nel caso di ricerche per tipo, cercare ad esempio “defense” ritornerà tutti i tipi che hanno le qualità di defense, ovvero defense, master ed evolution; cercare invece “master” ritornerà solamente il tipo master.

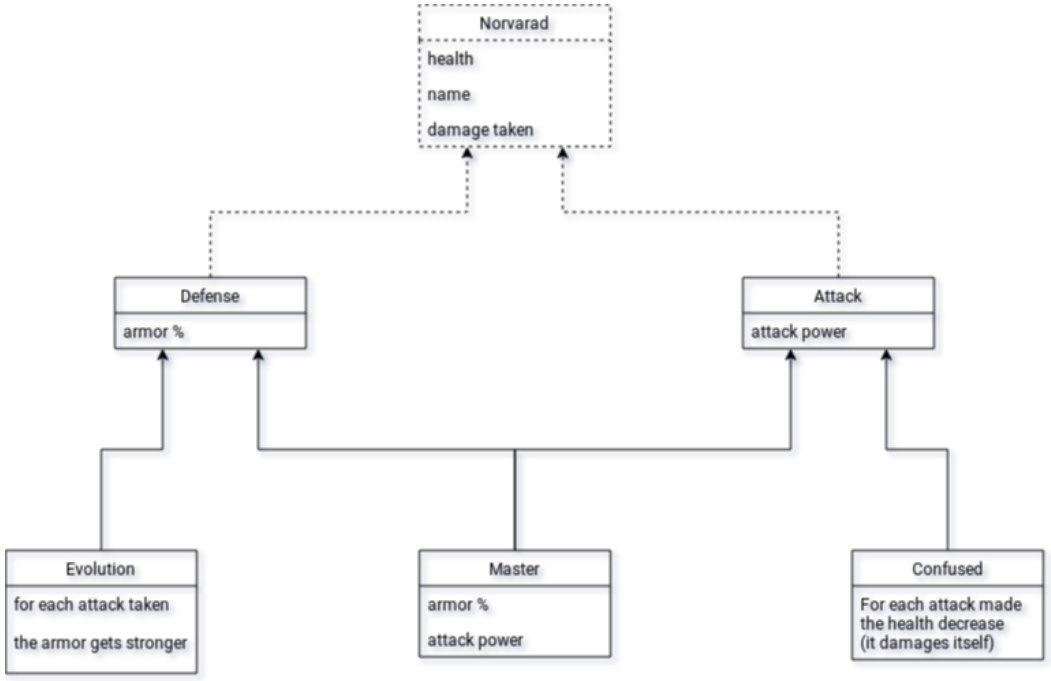
Per gli inserimenti: Selezionare il tipo desiderato e premere “Insert”, si aprirà una finestra pop-up dove inserire i dati necessari a creare il tipo selezionato.

Tramite il menù file è possilbile caricare e salvare su file usando le funzioni “Open” e “Save”, sempre nel menu’ sono indicate le varie shortcut da tastiera disponibili.

Nota: il tema grafico dell’applicazione si basa su quello del sistema operativo e quindi potrebbe variare da sistema a sistema.

Progettazione

Gerarchia di classi



La gerarchia è realizzata includendo ereditarietà a diamante e derivazioni virtuali. In particolare la calsse “Norvarad” è una classe base astratta con 5 sottoclassi istanziabili, tra cui la classe “Master” realizza l’ereditarietà a diamante. Ogni classe ha delle caratteristiche uniche rispetto alle altre, in particolare:

- Defense: possiede un valore di armatura in percentuale (<=85%), che permette di mitigare gli attachi subiti. Non può però effettuare attacchi.
- Attack: possiede potenza d’attacco e può quindi effettuare attacchi. Non ha armatura.
- Master: possiede le caratteristiche sia del tipo difesa che del tipo attacco, ovvero ha un armatura ma può anche effettuare attacchi.
- Evolution: per ogni attacco subito la sua armatura si migliora di un valore percentuale.
- Confused: per ogni attacco effettuato subisce parte del danno inflitto, ovvero per ogni attacco diminuisce la sua stessa salute.

In quanto a **polimorfismo**, la classe base astratta “Norvarad” prevede i metodi virtuali:

- `virtual std::string printDescription() const;` restituisce una stringa che descrive l’oggetto, riportandone quindi tutte le proprietà;
- `virtual void toXml(QXmlStreamWriter& writer) const;` simile alla funzione precedente, ma salva tutte le proprietà dell’oggetto su un file xml usando il QxmlStreamWriter passato come parametro;

- `virtual Norvarad& operator = (const Norvarad&) = default;` confronta due oggetti per decidere se sono equivalenti o no, di norma confronta il “campo value” degli oggetti ma è virtuale così che i sottotipi possano decidere diversi metodi di confronto;
- `virtual ~Norvarad() = default;`

che possono quindi essere sovrascritti dalle sottoclassi, ed i seguenti metodi virtuali puri che devono essere implementati nelle sottoclassi:

- `virtual std::string getType() const = 0;` ritorna una stringa col tipo dell’oggetto instanziabile;
- `virtual Norvarad* clone() const = 0;` ritorna un puntatore ad una copia profonda dell’oggetto di invocazione;
- `virtual double getValue() const = 0;` ritorna il valore assegnato ad ogni oggetto, che varia in base alle proprietà stesse dei singoli oggetti;
- `virtual percentage getArmor() const = 0;` ritorna la percentuale di armatura di un oggetto, valore che potrebbe cambiare da tipo a tipo sia come metodi di calcolo che come effettiva presenza;
- `virtual double getAttackPower() const = 0;` ritorna la potenza d’attacco dell’oggetto, vale lo stesso ragionamento del punto precedente;
- `virtual double takeDamage(double) = 0;` subisce un attacco aumentando quindi i danni subiti, ogni classe ha una diversa implementazione di questo metodo;
- `virtual double makeAttack() = 0;` Effettua un attacco, ogni classe ha il suo modo di effettuare un attacco (ad esempio la classe “Confused” diminuisce anche il valore di salute dell’oggetto di invocazione).

Ogni oggetto della gerarchia, per scelta progettuale e logica, prevede che possa essere modificato arbitrariamente solamente il suo nome tramite il metodo void **setName**(std::string n);. Gli altri parametri sono caratteristici dell’oggetto e possono essere modificati solamente dalle operazioni di attacco e difesa.
Ogni classe può possedere inoltre opportuni campi statici e costanti che ne descrivono i principali valori o attributi fissi ed immutabili, tali valori a seconda dei casi sono stati dichiarati privati o pubblici.

Contenitore

É stato inoltre realizzato, come da requisiti, un contenitore tramite template ed una classe di puntatori “smart” **DeepPtr**. Nella realizzazione del contenitore ho scelto di utilizzare una lista doppiamente linkata, principalmente perché nell’uso ordinario è previsto un uso frequente di rimozioni in punti casuali del contenitore stesso e, per quanto questi vantaggi siano mitigati dalla struttura di Qt (quasi sempre non si ha l’iteratore diretto ma bensì un indice), si è deciso comunque per questa implementazione in quanto ritenuta la più adeguato al caso d’uso, senza quindi vincolare questa scelta allo specifica libreria grafica utilizzata. Il contenitore conterrà oggetti di tipo DeepPtr contenenti a loro volta puntatori polimorfi Norvarad*.

Libreria Qt e GUI

Nel realizzare l’interfaccia grafica si è scelto di usare il pattern Model-View così da sfruttare appieno gli strumenti forniti dalla libreria Qt e velocizzare quindi lo sviluppo, separando comunque l’implementazione del modello logico dalla GUI.

Nel realizzare l’interfaccia grafica è risultato opportuno rappresentare il cotenitore tramite QlistView fornita da Qt, combinata ad un ProxyModelAdapter derivato da QsortFilterProxyModel (offerto quindi da Qt) e un ListModelAdapter derivato da QabstractListModel (sempre offerto da Qt) che si occupano di mantenere aggiornata la vista rispecchiando le modifiche effettuate al contenitore. Questa scelta ha permesso inoltre di effettuare ricerche nel container con visualizzazione aggiornata dinamicante e di sfruttare gli strumenti offerti da Qt per fare il match fra indici reali del contenitore ed indici visualizzati, così da permettere sempre rimozioni e modifiche anche sui sottoinsiemi di elementi ottenuti dalle operazioni di ricerca.

L’adapter ed in genere le classi GUI implementano tutti i metodi necessari all’utilizzo della grafica stessa e del contenitore, mantenendo però sempre una separazione dal vero modello logico dei dati così da permettere aggiornamenti grafici più semplici e per facilitare anche un’eventuale migrazione verso una nuova libreria grafica.

Salvataggio su file

Tramite l’apposito menù “File” (oppure con Ctrl+O e Ctrl+S) è possibile salvare su file e caricare la configurazione del container su file xml. La struttura interna del file è volutamente molto semplice e non sono previsti eccessivi controlli sulla struttura stessa del file poiché esso non deve essere modificato manualmente ma solo dal programma, e l’integrità dei dati in scrittura è garantita dall’uso della classe QSaveFile e dal metodo commit(). La scelta del file è resa disponibile da un QFileDialog impostato per permettere la selezione di soli file xml.

Per il salvataggio su file è stato implementato un metodo virtuale nella gerarchia di classi che si occupa, per ogni tipo, di salvare gli opportuni valori nel file xml. Nel caricamento si è scelto invece di leggere direttamente il file xml e di ricostruire gli oggetti usando i loro costruttori pubblici.

Esempio di struttura del documento:

```
<norvarads>
  <defense>
    <health>555</health>
    <damageTaken>0</damageTaken>
    <name>pippo</name>
    <armorLevel>45</armorLevel>
  </defense>
</norvarads>
```

Conclusioni ed estensibilità

Nella realizzazione del progetto si è sempre cercato di favorire le possibilità di sviluppo future, dal container alla gerarchia fino all'interfaccia grafica. Nelle classi sono presenti metodi creati per implementazioni che poi non hanno trovato il tempo di essere già realizzate, ma che potrebbero rappresentare nuove funzionalità (operatori, funzioni di stampa, simulazioni combattimenti etc.). Inoltre, la modifica dell'interfaccia grafica può avvenire senza richiedere modifiche al modello e viceversa, si potrebbe ad esempio cambiare l'implementazione del container passando ad un array dinamico senza dover aggiornare la parte grafica ed ovviamente senza modificare la gerarchia. Per aggiungere nuovi tipi alla gerarchia basta estendere quelli esistenti e modificare in modo opportuno la rappresentazione nella lista e la funzione di load da file.