Università degli Studi di Padova

Laurea in Informatica

WoWDB
World of Warcraft Database

Nicola Zorzo
1122259

Giugno 2019

Relazione sul progetto di Programmazione ad Oggetti a tema "Qontainer".

A.A. 2018-2019

1 Introduzione

WoWDB è un'applicazione ispirata all'MMORPG (*Massive Multiplayer Online Role-Playing Game*) *World of Warcraft* ed ha lo scopo di simulare un *container* in cui vengono gestiti i personaggi che appartengono a tale universo.

Le principali funzionalità offerte sono:

- inserimento, modifica e rimozione personaggi;
- ricerca e ordinamento personaggi;
- salvataggio e caricamento di *container*.

2 Progettazione

Durante la fase di progettazione sono stati presi in considerazione i seguenti punti, con le relative soluzioni adottate:

- gestione automatica della memoria: implementazione di un template di classe `DeepPtr<T>` di puntatori polimorfi *smart* al tipo `T`, come consigliato;
- accesso frequente agli elementi del *container* in posizioni arbitrarie: implementazione di un template di classe `Container<T>` come array dinamico, piuttosto che come lista.

Il *design pattern* utilizzato è il **Model-View**, per i seguenti motivi:

- separazione chiara tra modello logico e GUI;
- ampio supporto per questo tipo di architettura da parte di Qt;
- monte di ore a disposizione limitato.

3 Descrizione gerarchia

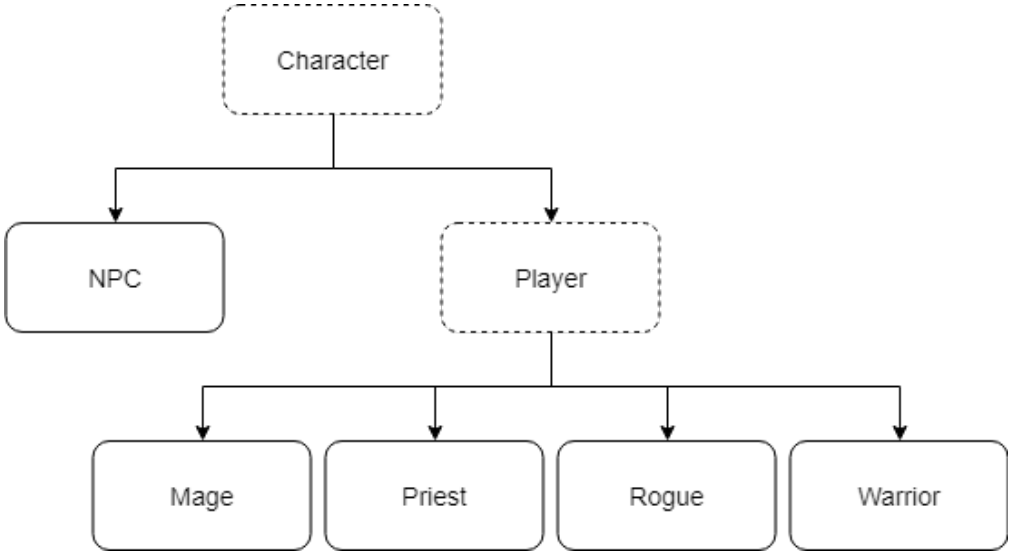

Figura 1: Gerarchia delle classi

La gerarchia è composta dalle seguenti classi:

- *Character*: classe base astratta polimorfa che rappresenta le caratteristiche comuni a tutti i personaggi, quali:
  - nome;
  - sesso;
  - razza;
  - fazione;
  - livello;
  - punti vita;
- *NPC*: classe concreta derivata da *Character* che rappresenta tutti i *Non-Player Character*, ossia i personaggi che non sono controllati da un giocatore ma dal gioco, con i seguenti campi:
  - tipo di *NPC*;
  - tipo di oggetti in vendita;
  - presenza di una *quest*;

- *Player*: classe astratta derivata da *Character* che rappresenta tutti i personaggi controllati da un giocatore, con i seguenti campi:
  - stato (online oppure offline);
  - *rating* (punteggio del personaggio in *Player vs Player*);
  - statistiche di base (forza, agilità, intelletto, spirito);
- *Mage, Priest, Rogue, Warrior*: classi concrete derivate da *Player*, che rappresentano in modo effettivo la *classe* che il giocatore ha deciso di utilizzare quando crea un personaggio, con le proprie caratteristiche. Queste classi non hanno campi aggiuntivi, si limitano a implementare i metodi astratti forniti dalla superclasse e ne forniscono di nuovi.

4 Polimorfismo

La classe base *Character* fornisce i seguenti metodi virtuali pubblici:

- `virtual ~Character() = default`: distruttore virtuale alla base della gerarchia per permettere la corretta invocazione del distruttore nelle classi derivate;
- `virtual string getClass() const = 0`: metodo d'utilità che restituisce una stringa identificativa della classe di appartenenza di un oggetto;
- `virtual Character* clone() const = 0`: metodo d'utilità che crea una copia profonda dell'oggetto d'invocazione, obbligatorio per il supporto dell'implementazione del template di classe *DeepPtr* per la gestione automatica della memoria;
- `virtual QJsonObject serialize() const`: metodo utilizzato per la serializzazione in JSON di un oggetto della gerarchia;
- `virtual void deserialize(const QJsonObject& obj)`: metodo complementare al precedente, utilizzato per la costruzione di un oggetto della gerarchia a partire da un oggetto in formato JSON.

La classe derivata concreta *NPC* non fornisce metodi polimorfi e si limita a implementare i metodi della classe base.

La classe derivata astratta *Player* aggiunge ai metodi virtuali della classe base i seguenti metodi:

- `virtual double strengthMultiplier() const = 0`: metodo che restituisce il fattore moltiplicativo per l'attributo forza;
- `virtual double agilityMultiplier() const = 0`: metodo che restituisce il fattore moltiplicativo per l'attributo agilità;
- `virtual double intellectMultiplier() const = 0`: metodo che restituisce il fattore moltiplicativo per l'attributo intelletto;
- `virtual double spiritMultiplier() const = 0`: metodo che restituisce il fattore moltiplicativo per l'attributo spirito;
- `virtual int attackPower() const`: metodo che restituisce il potere d'attacco fisico di una classe, calcolato in base alle statistiche di base, al loro fattore moltiplicativo e, dove presente, a particolarità della classe stessa;
- `virtual int spellPower() const`: come il metodo precedente, ma restituisce il potere d'attacco magico di una classe;
- `virtual string armor() const = 0`: metodo che restituisce il tipo di armatura utilizzata;
- `virtual string role() const = 0`: metodo che restituisce una stringa contenente i ruoli che può ricoprire una determinata classe.

Le classi derivate concrete *Mage, Priest, Rogue, Warrior* non forniscono ulteriori metodi polimorfi e, come *NPC*, si limitano a implementare tutti i metodi virtuali che hanno ereditato. Ciò non esclude la possibilità di introdurne di propri per l'eventuale estensione di una specifica classe.

5 JSON

Il formato scelto per i file di salvataggio/caricamento dei *container* è JSON. Questo formato ha il vantaggio di essere facilmente comprensibile dalle persone e permette una manipolazione molto agevole dei dati, sia per il processo di *serializzazione* (da istanza di una classe a oggetto JSON) che per il processo inverso di *deserializzazione* (da oggetto JSON a istanza di una classe). Inoltre Qt fornisce una ricca API per il *parsing*, la modifica e il salvataggio di questo tipo di dati.

Di seguito sono riportati degli esempi di JSON ottenuti dopo la serializzazione di un'istanza delle classi *NPC* e *Rogue*, invocando il metodo `serialize()` precedentemente descritto.

```
{
    "class": "NPC",
    "faction": "Horde",
    "gender": "Male",
    "hp": 100000,
    "level": 85,
    "merchandise": "None",
    "name": "Garrosh",
    "quest": "Slaughter the Alliance!",
    "race": "Orc",
    "type": "Boss"
},
{
    "agility": 1620,
    "class": "Rogue",
    "faction": "Alliance",
    "gender": "Male",
    "hp": 10000,
    "intellect": 32,
    "level": 80,
    "name": "Kalimist",
    "race": "Human",
    "rating": 2200,
    "spirit": 15,
    "status": "Online",
    "strength": 120
}
```


6 Suddivisione oraria

Il tempo impiegato per la realizzazione del progetto è stato di circa 50 ore, suddivise come segue:

- analisi del problema: 2 ore;
- progettazione modello: 3 ore;
- progettazione GUI: 3 ore;
- apprendimento libreria Qt: 8 ore;

- codifica modello: 7 ore;
- codifica GUI: 22 ore;
- debugging: 3 ore;
- testing: 2 ore.

7 Ambiente di sviluppo

L'ambiente utilizzato per sviluppare il progetto è il seguente:

- sistema operativo: Ubuntu 19.04;
- versione compilatore GCC: 8.3.0;
- versione Qt Creator: 5.12.

Il progetto è stato in seguito testato sulla macchina virtuale fornita e sui computer dei laboratori LabTA e P036.

8 Compilazione ed esecuzione

Il progetto prevede l'utilizzo di un file `.pro` diverso da quello generato attraverso il comando `qmake -project` a causa della presenza di diverse funzionalità introdotte a partire da C++11.
Di conseguenza, per compilare ed eseguire il progetto, basta raggiungere da terminale la cartella `WoWDB` e lanciare i seguenti comandi:

1. `qmake`
2. `make`
3. `./WoWDB`

Per testare subito tutte le funzionalità offerte dall'applicazione, viene fornito un *container* di prova denominato `test.json` presente all'interno della cartella `WoWDB/data`.

9 Note aggiuntive

World of Warcraft è un gioco con varie espansioni e quella presa come riferimento è *Wrath of the Lich King*. Per ricreare un ambiente che rispecchia in modo adeguato tale espansione è stato posto un *range* di valori accettabili per i vari campi dati, che vengono corretti in automatico qualora non fossero rispettati:

- `level: [1, 80]`;
- `HP: [1, 999999]`;
- `rating: [0, 4000]`;
- `strength, agility, intellect, spirit: [0, 2000]`.

Inoltre, per ogni fazione presente (`Alliance, Horde, Neutral`), è possibile creare solamente dei personaggi la cui razza appartiene a quella fazione. Gli accoppiamenti permessi sono:

- `Alliance: Draenei, Dwarf, Gnome, Human, Night Elf`;
- `Horde: Blood Elf, Orc, Tauren, Troll, Undead`;
- `Neutral`: tutte le razze possono appartenere alla fazione neutrale.