**oggetto** Relazione progetto Programmazione ad Oggetti

**gruppo** Meneghini Fabio, mat. 2034332

**titolo** Fantasy Battle Simulator

# Introduzione

Fantasy Battle Simulator, come suggerisce il nome, è un semplice videogioco che permette di simulare delle battaglie tra due team composti da personaggi fantasy. Esso permette di creare i due team aggiungendo, modificando e cancellando i personaggi, con la possibilità di salvare i team su file JSON, e viceversa caricarli da file JSON. I personaggi disponibili sono di diversa tipologia: sono presenti guerrieri, stregoni, guaritori, scheletri e goblin, e ognuno attacca il team avversario in modo diverso o calcolando il danno in modo diverso.

Il principale punto di forza di questo progetto è la gestione delle battaglie a turni, realizzata da zero, prendendo come esempio le battaglie di alcuni videogiochi reali, come i vecchi giochi Pokémon o i vecchi Final Fantasy. Sebbene piuttosto minimale, sono state implementate le principali funzionalità, ad esempio la gestione della morte di uno o più membri del team, la gestione della vittoria o della sconfitta, la possibilità di cambiare personaggio da utilizzare durante la battaglia e la possibilità di arrendersi.

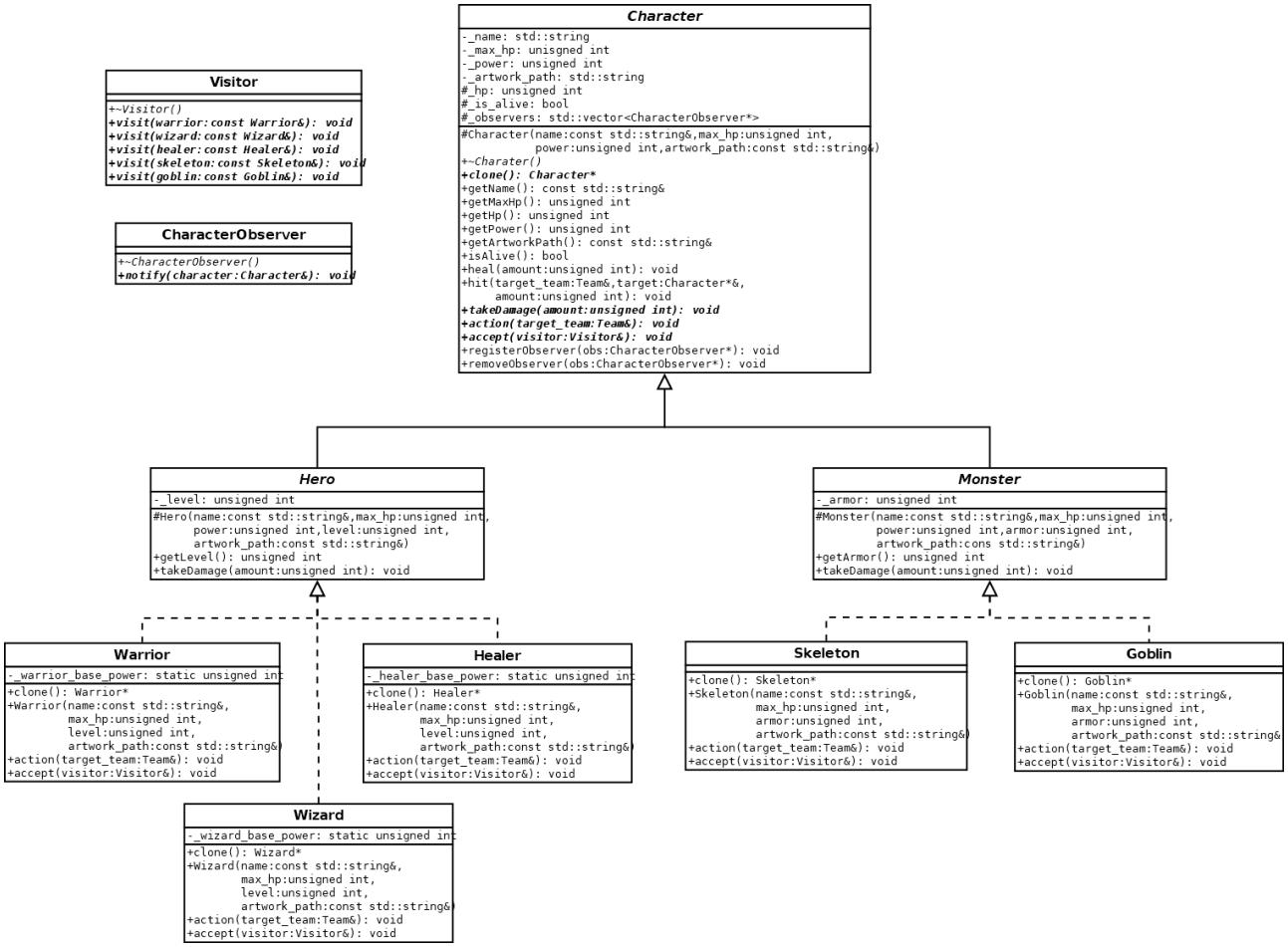
Ho scelto questo progetto perché sono appassionato di videogiochi e ho un po' di esperienza nello sviluppo di giochi 2D con motori grafici come Unity e Godot, perciò ho voluto prendere l'occasione per provare ad utilizzare uno strumento più di "basso livello" per realizzare un semplice prototipo di videogioco. Inoltre ho pensato che i diversi tipi di personaggi disponibili potessero essere un buon pretesto per costruire widget grafici leggermente diversi sulla base del tipo concreto del personaggio in questione, realizzando così il polimorfismo non banale richiesto nelle specifiche.

# Descrizione del modello

Il modello logico è composto dalla gerarchia di classi che rappresenta i vari tipi di personaggi utilizzabili (di cui è riportato in seguito il diagramma UML), dal team, cioè il contenitore per le classi dei personaggi, e da altre classi per implementare i design pattern *Visitor* e *Observer*.

Il modello parte da una classe astratta *Character* che contiene le informazioni comuni a tutti i personaggi, ovvero nome, punti vita massimi e attuali, potenza, percorso dell'artwork, un booleano che indica se il personaggio è vivo o meno e un vettore che contiene gli Observer del personaggio. Per ciascuno di questi attributi sono stati implementati dei metodi *getter*. Sono inoltre presenti metodi per la gestione delle battaglie, in particolare *takeDamage* e *action* sono virtuali puri, in quanto la loro implementazione varia in base al tipo concreto del personaggio. La sotto-gerarchia *Hero* rappresenta i personaggi "eroi", che sono caratterizzati da un attributo aggiuntivo che rappresenta il loro livello di esperienza, usato nel calcolo del danno inflitto all'avversario, e si articolano nelle tre classi concrete *Warrior*, *Wizard* e *Healer*. Ciascuno di questi tre tipi di personaggi offre la sua implementazione del metodo *action*: in particolare, il guerriero attacca il primo personaggio del team passato come parametro al metodo, lo stregone attacca tutti i membri del team e il guaritore cura tutti i membri del team che non sono sottotipi di *Monster*. Nel calcolo del danno viene tenuto conto della potenza base della classe in questione, implementato come una costante di classe, cioè come campo dati statico con valore differente per ciascuna classe, e solo per gli "eroi" anche del loro livello di esperienza.

Dualmente, la sotto-gerarchia *Monster* rappresenta i personaggi “mostri”, che a differenza dagli “eroi” sono caratterizzati da un attributo numerico che rappresenta la resistenza della loro armatura, usato nel calcolo del loro danno ricevuto, e si articola nelle due classi concrete *Skeleton* e *Goblin*, ciascuna delle quali implementa il metodo *action* in modo diverso. In particolare, lo scheletro attacca tutti i membri del team avversario, mentre il goblin solo il primo.




Vi è inoltre la sezione relativa al *Team*, che si comporta come classe contenitore per le classi che fanno parte della gerarchia *Character*. La classe *Team* è fondamentalmente un arricchimento di un array di puntatori a *Character*: tale classe è caratterizzata, oltre all’effettivo array dinamico di puntatori a *Character*, dalla dimensione del team, cioè dal numero di elementi che sono stati effettivamente inseriti, dal numero di personaggi ancora in vita contenuti al suo interno (utile durante le battaglie) e dalla sua capacità, cioè il numero massimo di elementi che può contenere (utile soprattutto per non dover riallocare il Team con la dimensione aggiornata ad ogni inserimento). Oltre ai metodi per aggiungere, rimuovere o cercare un elemento, sono presenti anche dei metodi utili alla gestione delle battaglie: in particolare, *moveBack* sposta in coda il personaggio puntato dal puntatore a *Character* passato come parametro, e viene usato quando tale personaggio viene sconfitto; *leftShiftAlive*, invece, serve per gestire il cambio del personaggio attivo (cioè quello effettivamente controllato dall’utente) durante una battaglia (come suggerisce il nome, l’operazione che viene fatta è un left shift tra i personaggi non sconfitti). Tali metodi modificano solo il contenuto del team e non la vera e propria visualizzazione grafica: a tale fine viene utilizzato il design pattern *Observer* (sia per aggiornare la visualizzazione dei parametri del personaggio attualmente attivo, sia per aggiornare l’ordine degli artwork dei personaggi che compongono il team).

## Polimorfismo

In questo progetto l'utilizzo principale del polimorfismo riguarda il design pattern *Visitor* nella gerarchia *Character*. Esso viene usato per la conversione in formato JSON per realizzare la persistenza dei dati e per costruire dei widget grafici che mostrano informazioni diverse a seconda del tipo concreto dell'oggetto a cui si riferiscono (per esempio viene usato nel pannello che mostra i parametri e le informazioni riguardanti il personaggio attivo durante le battaglie).

Vi è anche un utilizzo del polimorfismo di minor rilevanza: vista l'esigenza di un "costruttore di copia polimorfo" viene usato il design pattern *Clone* per simularne l'effetto, inoltre, come già descritto precedentemente, la classe *Character* dichiara dei metodi viruali puri per la gestione delle battaglie, cioè *takeDamage*, le cui implementazioni (diverse) sono fornite dalle classi *Hero* e *Monster*, e *action*, le cui implementazioni (diverse) sono fornite da ciascuna classe concreta di tale gerarchia.

# Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato JSON. Viene usato un unico file per memorizzare due vettori di personaggi che rappresentano due team (uno degli alleati e uno dei nemici). I personaggi sono rappresentati nel file tramite associazioni chiave-valore dei loro attributi, aggiungendo un campo *type* che serve per memorizzare l'informazione sul tipo concreto del personaggio in questione. Un esempio sulla struttura dei file è dato dal file JSON fornito assieme al codice (denominato *file.json*), che contiene due team già pronti in modo da poter provare le varie funzionalità del progetto.

# Funzionalità implementate

Le funzionalità implementate sono le seguenti:

- conversione e salvataggio dei team in formato JSON
- possibilità di svolgere battaglie tra i due team creati
- possibilità di cambiare personaggio da utilizzare durante una battaglia
- possibilità di arrendersi durante una battaglia
- gestione della sconfitta di uno o più membri del team
- gestione dell'esito della battaglia (vittoria / sconfitta)

Le funzionalità grafiche, invece, sono le seguenti:

- barra dei menù in alto
- utilizzo di icone nelle voci del menù
- status bar in fondo alla finestra
- scorciatoie da tastiera (mostrate anche nelle voci del menù)
- chiede conferma prima di uscire dal programma o prima di arrendersi da una battaglia
- controlli che assicurano che l'utente possa premere i tasti / voci del menù solo nei momenti opportuni, con relativi avvisi / messaggi di errore
- gestione di più schermate

- utilizzo di immagini nella visualizzazione dei personaggi durante le battaglie
- cambio dell’immagine di un personaggio quando viene sconfitto
- utilizzo di colori e stili grafici
- effetti grafici come cambio del colore al passaggio del mouse

Le funzionalità elencate sono intese in aggiunta a quanto richiesto dalle specifiche del progetto.

# Rendicontazione ore

| Attività | Ore Previste | Ore Effettive |
|---|---:|---:|
| Studio e progettazione | 10 | 8 |
| Sviluppo del codice del modello | 10 | 11 |
| Studio del framework Qt | 10 | 9 |
| Sviluppo del codice della GUI | 10 | 12 |
| Ricerca e editing degli assets | 1 | 2 |
| Test e debug | 5 | 9 |
| Stesura della relazione | 4 | 4 |
| **totale** | 50 | **55** |

Il monte ore è stato leggermente superato in quanto l’implementazione delle battaglie ha richiesto più tempo del previsto, in particolare ho speso molto tempo in test e debug per risolvere alcuni errori a run-time di cui inizialmente non mi ero accorto.