Progetto di Programmazione ad Oggetti

A.A. 2019/2020

Pavin Nicola - Matricola: 1193215
Sertori Nicholas - Matricola: 1193171

FightingGame

Relazione di Nicholas Sertori

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

Abstract

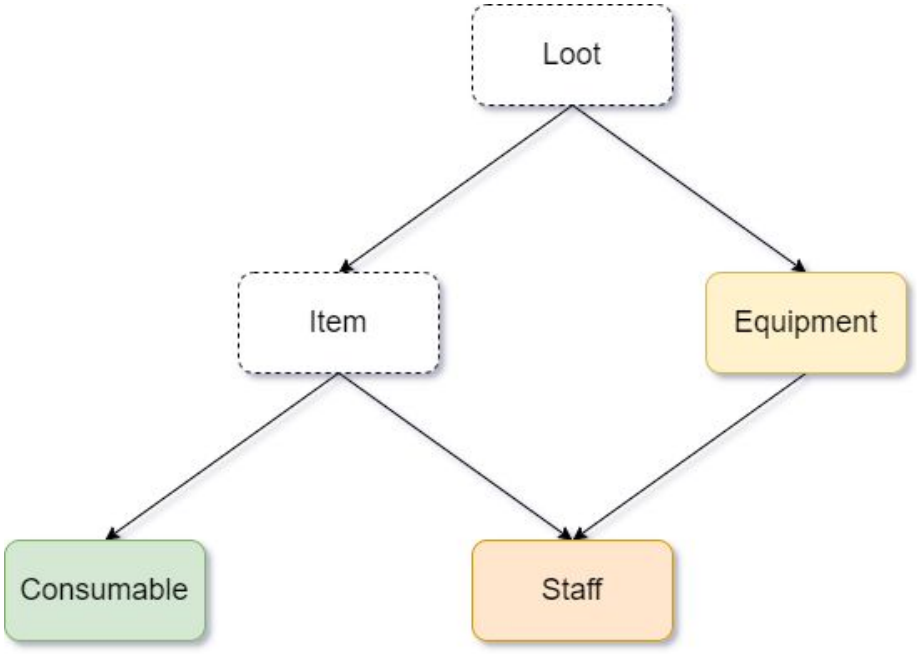
L'idea dietro a questo progetto è stata quella di creare un semplice videogioco, tanto semplice che è forse esagerato chiamarlo "videogioco", che rappresentasse in modo basilare i combattimenti di un generico gioco di ruolo.
Volevamo un'interfaccia semplice che ci permettesse di interagire con un "nemico" attraverso attacchi all'arma bianca, e ci interessava inserire l'uso di un semplice inventario il cui scopo all'interno del gameplay è quello di rendere più vario il combattimento, perchè fosse meno monotono e lineare.
Già dall'inizio era presente inoltre l'idea di poter affrontare più nemici in successione.

Progettazione di FightingGame

Il progetto è modellato secondo una struttura "Model-View-Controller", tenendo separata la parte logica dalla GUI. Facciamo notare che all'interno del modello e della gerarchia di classi che compongono lo scheletro del progetto è stato fatto uso di classi appartenenti alla libreria Qt, come QFile e QString, per facilitare l'uso delle risorse esterne (file .txt presi in input per la generazione di determinati oggetti che verranno descritti in seguito). Questo non influisce tuttavia nella separazione tra parte logica e GUI, poichè le classi Qt inserite nel modello non aggiungono nulla a livello di interfaccia, ma soltanto funzionalità progettuali.

Gerarchia di Classi

La parte logica del progetto che ne costituisce lo scheletro è formata da una gerarchia di classi che presenta ereditarietà multipla.
Le classi coinvolte sono strutturate come segue:



L'idea è quella di rappresentare gli oggetti che si possono trovare all'interno del gioco utilizzando questa gerarchia per facilitarne la gestione e l'inventariazione.

La classe Loot incapsula i dati più generici degli oggetti di gioco, ovvero nome e ID. Essa dichiara inoltre alcune delle funzioni che le classi successive reimplementano o richiamano nella gestione dei propri sotto-oggetti.
Le classi Item ed Equipment si differenziano nello scopo che gli oggetti racchiusi in esse devono svolgere. Mentre gli Equipment sono armi e armature, di cui il giocatore potra possedere e indossare un solo elemento per tipo (armor e weapon), gli Item sono gli oggetti nell'inventario, utilizzabili durante il combattimento per potenziarsi o indebolire l'avversario.
La classe Consumable rappresenta l'implementazione effettiva di Item, con l'aggiunta di un campo che rappresenta gli usi rimasti all oggetto prima che sia consumato.
La classe Staff ha derivazione multipla ed eredita sia le caratteristiche di Item, come effetti che si applicano sulle caratteristiche di un Character, sia le caratteristiche di Equipment, come il valore base dell'arma e i suoi usi rimasti, ovvero la sua condizione (durability).
Le Staff rappresentano quindi bastoni o bacchette, o catalizzatori magici di un qualche tipo, che possono essere utilizzati sia per attaccare normalmente, impugnandoli come arma, sia per "castare" effetti sulle caratteristiche proprie o dell'avversario, come succede per gli Item.

Oltre alle classi Model e Controller, che assieme alla vista contribuiscono alla struttura del progetto aderendo al modello MVC, abbiamo implementato una classe Character che non fa parte della gerarchia sopra menzionata, e che gestisce tutto ciò che riguarda il giocatore (o l'avversario che pur essendo guidato dal modello e dal controller nelle sue azioni, è a tutti gli effetti un personaggio).

È inoltre presente, come richiesto da progetto, una classe contenitore template chiamata sList che gestisce l'inventario di ogni personaggio, e che non è altro che una lista single-linked, corredata di operatori e funzioni adatte al nostro progetto ed altre aggiunte per renderla completa e utile anche in altri ambiti.

Chiamate Polimorfe

Grazie alla gerarchia delle classi è stato possibile creare varie chiamate a funzioni polimorfe che semplificano la scrittura e il funzionamento del progetto.

- ***powerUp()***

Questa funzione si occupa del potenziamento degli item.
Terminata una battaglia, veniamo premiati con degli item che vengono inseriti nel nostro inventario. Nel caso uno di questi item dovesse essere già presente nel nostro inventario, verrà potenziato.

La chiamata polimorfa permette di gestire l'oggetto in modo corretto, che esso sia un consumable o una staff (classi che derivano dalla classe item), senza fare uso di typechecking. In base al tipo dell'oggetto verrà chiamata una *powerUp()* specifica che si occuperà dei campi propri, e di chiamare la *powerUp()* generica di Item che gestisce le modifiche ai campi "Health", "Attack" e "Defence".

- ***assign(ss)***

Questa funzione si occupa del parsing e della assegnazione di oggetti a partire da uno stringstream.
Ogni volta che viene costruito un oggetto appartenente alla gerarchia di classi presentata prima, prendendo come input una linea di testo proveniente da un file di input (.txt), la funzione *assign(ss)* si occupa di tradurre tale linea di testo in dati utili alla costruzione dell'oggetto in questione.
A tale scopo la funzione assign è polimorfa e ridefinita (ampliata se vogliamo) all'interno di ogni classe della gerarchia.
Le linee di testo sono scritte in modo da rispettare sempre un determinato ordine che permetta ad ogni sotto-classe di chiamare la sua classe base passandogli solo la parte di interesse della stringa di input, costruendo poi autonomamente i dati propri utilizzando il resto della stringa.

- ***getDesc()***

Questa funzione restituisce una panoramica dell'oggetto su cui è invocata. Non si limita a restituire il campo "description" qualora presente, ma aggiunge un insieme di dati presi dagli altri campi e tradotti in stringhe. Il risultato è appunto una panoramica più completa dell'oggetto in questione, utile ad esempio per essere messa a comparsa quando stazioniamo su un itemButton nell'inventario.
Il motivo per cui questa funzione è polimorfa è evidente se inserito nel contesto della nostra gerarchia di oggetti con descrizioni e funzioni differenti.

- ***effect()***

Questa funzione effettua l'azione caratteristica dell'oggetto di invocazione, che può essere dinamicamente un consumable o uno staff. In sostanza non fa che ritornare l'effetto dell'item, ovvero i modificatori che applica a "health", "attack", e "defence", apportando poi determinate modifiche all'oggetto di invocazione, in base alla classe a cui questo appartiene.

- ***getRemainingUses()***

Come per la funzione precedente, questa funzione si può applicare ad un consumable o ad una staff, e ritorna gli usi rimasti a disposizione per poterne impedire l'utilizzo all'interno dell'inventario una volta terminati.
Le due classi hanno modi diversi per calcolare questo dato, ma grazie alla funzione polimorfa non serve gestirlo separatamente.

- ***isUsable()***

Simile a getRemainingUses, lo scopo di questa funzione è controllare se il numero di usi rimasti dell'oggetto di invocazione è ancora positivo o se è sceso a zero.

- ***clone()***

Questo metodo permette di copiare in modo polimorfo gli oggetti della gerarchia di classi. Si è rivelato particolarmente utile nel passare un equipment dereferenziato da shared_ptr senza perdere questo dato una volta deallocati tutti i puntatori contati dallo shared_ptr.

File di Input

Per gestire al meglio la creazione di oggetti pseudo-randomici con cui premiare il giocatore, abbiamo deciso di costruire dei file (.txt) da usare come input.
Sono tre, e rappresenta le tre classi non astratte della gerarchia.
Vengono lette dal programma una linea alla volta, e per questo motivo la prima contiene soltanto il numero di linee successive, per poter lanciare un errore nel caso il numero di oggetti richiesti da creare sia superiore al numero effettivo di elementi presenti nel file. Ogni linea successiva rappresenta un singolo oggetto, riportando il contenuto di ogni suo campo.
Per esempio la creazione di un Equipment richiede una linea di testo simile alla seguente:

```
Spada_Generica 120 500 W 10
```

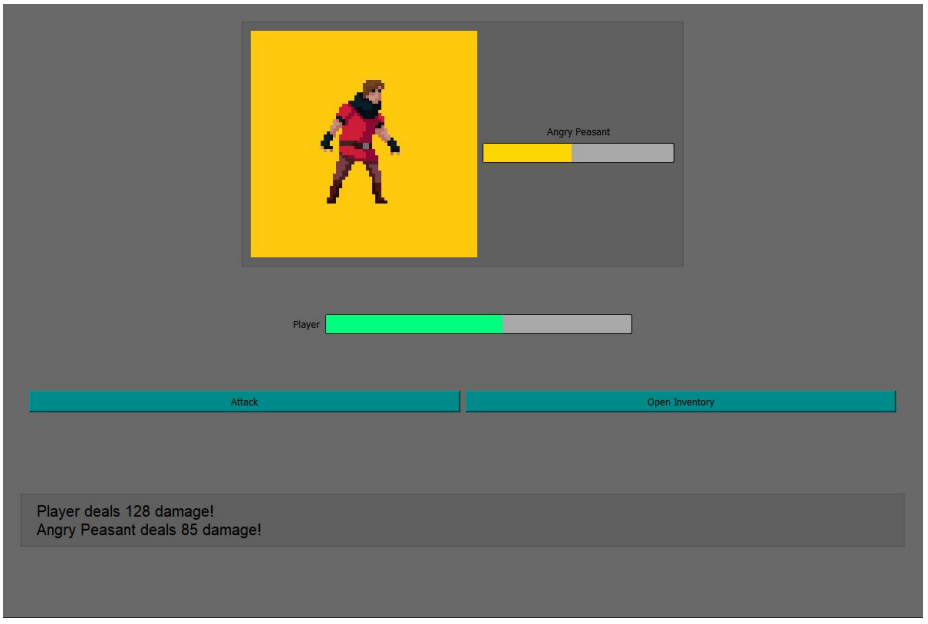
Dove ogni elemento separato dallo spazio è un campo dell'oggetto "equipment" che si vuole creare. I nomi e le descrizioni sono scritte con underscore al posto di spazi e vengono "normalizzati" in seguito, per garantire un corretto funzionamento dell'operatore di acquisizione >> che prende un segmento della stringa già inserita nello stream apposito.

Sono inoltre presenti delle risorse di tipo grafico (file .png) utilizzate per la costruzione della view.

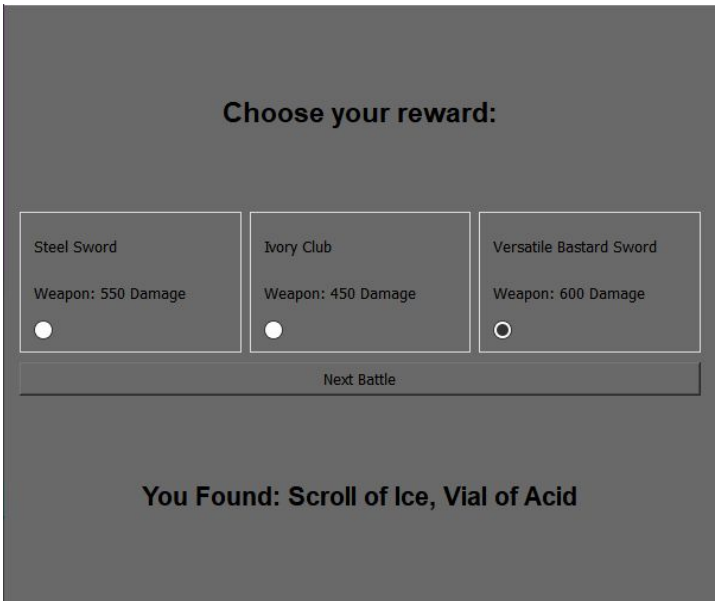
Manuale per l'utilizzo della GUI

Data la semplicità del gioco, la GUI è piuttosto lineare e di facile utilizzo.
Tutti i menu e le azioni di gioco sono gestite tramite bottoni.
Nella schermata di avvio è possibile accedere ad una breve descrizione del gioco che ne riassume lo scopo. Una volta inziata la partità ci verrà chiesto di inserire il nome del proprio personaggio, per poi dare il via alla battaglia.
Durante la battaglia potremo decidere di attaccare o accedere all'inventario per usare un item.

L’area sottostante i bottoni di azione serve a mostrare cosa succede durante il combattimento e aggiornarci riguardo a quale è stato l’effetto delle nostre azioni e di quelle dell’avversario.
All’interno dell’inventario possiamo vedere tutti i nostri item, e mantenendo il mouse sopra ai bottoni potremo vedere delle informazioni riguardo a ciò che gli item fanno prima di decidere se e quale usare. Da qui possiamo sempre decidere di tornare indietro e attaccare invece che usare un item, se quelli che abbiamo non ci soddisfano.
L’intera battaglia prosegue in questo modo fino alla sconfitta nostra o dell’avversario.
In caso di vittoria, prima di passare alla schermata successiva ci verrà chiesto di scegliere un equipaggiamento premio da sostituire a quello corrente (l’arma che impugnamo o l’armatura che indossiamo, o raramente una staff da poter impugnare e aggiungere all’inventario).
Una volta terminata una partita (vincendo o perdendo) saremo riportati al menù principale e si potrà immediatamente iniziarne una nuova.



Nell’immagine è possibile vedere i vari elementi della vista durante la fase di combattimento.



Ecco come si presenta il dialogo che mostra le reward a fine battaglia.

Sertori Nicholas

Compilazione ed Esecuzione

Per compilare ed eseguire il codice del progetto è strettamente necessario utilizzare il file (FightingGame.pro) fornito da noi che racchiude l'informazione necessaria alla corretta localizzazione dei file di input, delle informazioni sul gioco (how to play) e delle immagini.

Ambiente di Sviluppo

Il progetto è stato sviluppato su sistema operativo Windows 10. Abbiamo usufruito dell'IDE QtCreator in versione Qt:5.14.2. Mentre all'inizio il compilatore era MSVC_2017, siamo dovuti passare a MinGW 7.3.0 per un bug nella derivazione multipla a diamante.
Siamo infine passati alla macchina virtuale fornita per gli ultimi test di compatibilità.

Sviluppo del Progetto e Tempistiche

L'intero progetto è stato pensato e sviluppato in contemporanea da entrambi i membri del gruppo. Abbiamo lavorato tenendoci in contatto per consentire al progetto di mantenere una buona coesione.
L'ammontare di ore speso da ognuno per le varie mansioni risulta come segue:

1 ora - Analisi del problema e decisione progettuale.
15 ore - Ideazione e costruzione della gerarchia di classi e della classe Character.
10 ore - Ideazione e costruzione del Modello gestione Input.
5 ore - Costruzione del contenitore.
13 ore - Costruzione della View ì, comprese le ore per apprendere le librerie di Qt.
8 ore - Debugging e Testing/Calibrazione dei valori per equilibrare il combattimento.

In totale il progetto ha richiesto circa 52 ore di lavoro. Il superamento delle cinquanta ore richieste dal progetto è dovuto senza dubbio alla grande quantità di tempo necessaria a spulciare le librerie Qt in cerca dell'elemento giusto per personalizzare e costruire in modo corretto la Vista del progetto. Molto tempo è stato inoltre richiesto dalla gestione dei file di input, che hanno subito vari stadi e iterazioni diverse prima di raggiungere lo stato finale in cui sono stati consegnati, anche a causa di problemi con il riconoscimento di risorse esterne da parte di QtCreator.