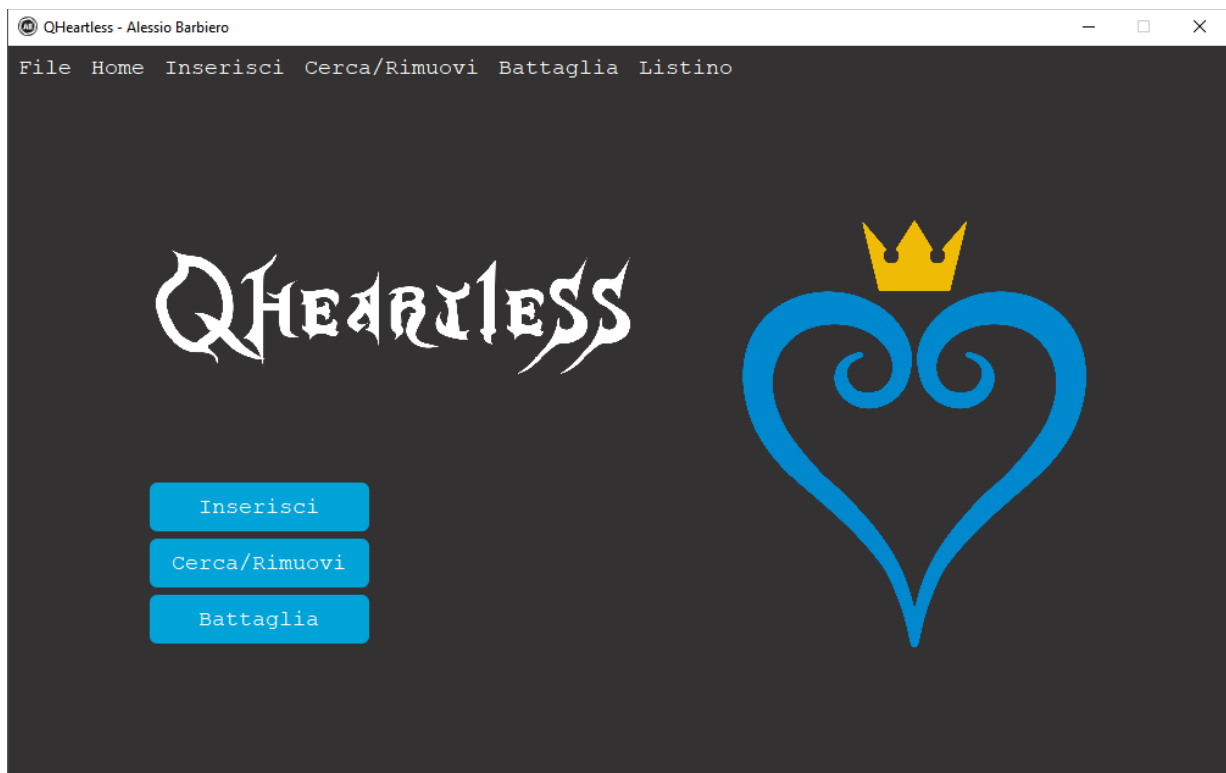


Progetto di programmazione ad oggetti – QHeartless



Descrizione del progetto

Il progetto realizza un container che consente di gestire, memorizzare ed interagire con gli Heartless, personaggi di finzione tratti dal videogioco Kingdom Hearts prodotto da Square Enix. Ogni Heartless è caratterizzato da una specie, che ne definisce le proprietà singolari, un nome e un valore di vita di partenza e un valore di vita attuale. Tramite l'interfaccia presentata è possibile inserire e rimuovere i personaggi all'interno del contenitore. È inoltre possibile simulare un attacco tra due diversi personaggi in modo da distinguerne le peculiarità in battaglia. Se un Heartless perde completamente la propria vita in seguito ad uno scontro, il corrispondente report viene evidenziato in rosso ed il suo utilizzo in battaglia non sortisce alcun effetto né in attacco né in difesa. Tramite un menù a barra è possibile spostarsi tra le varie finestre di interazione con il programma e salvare o caricare i dati dei personaggi (memorizzati come file .xml).

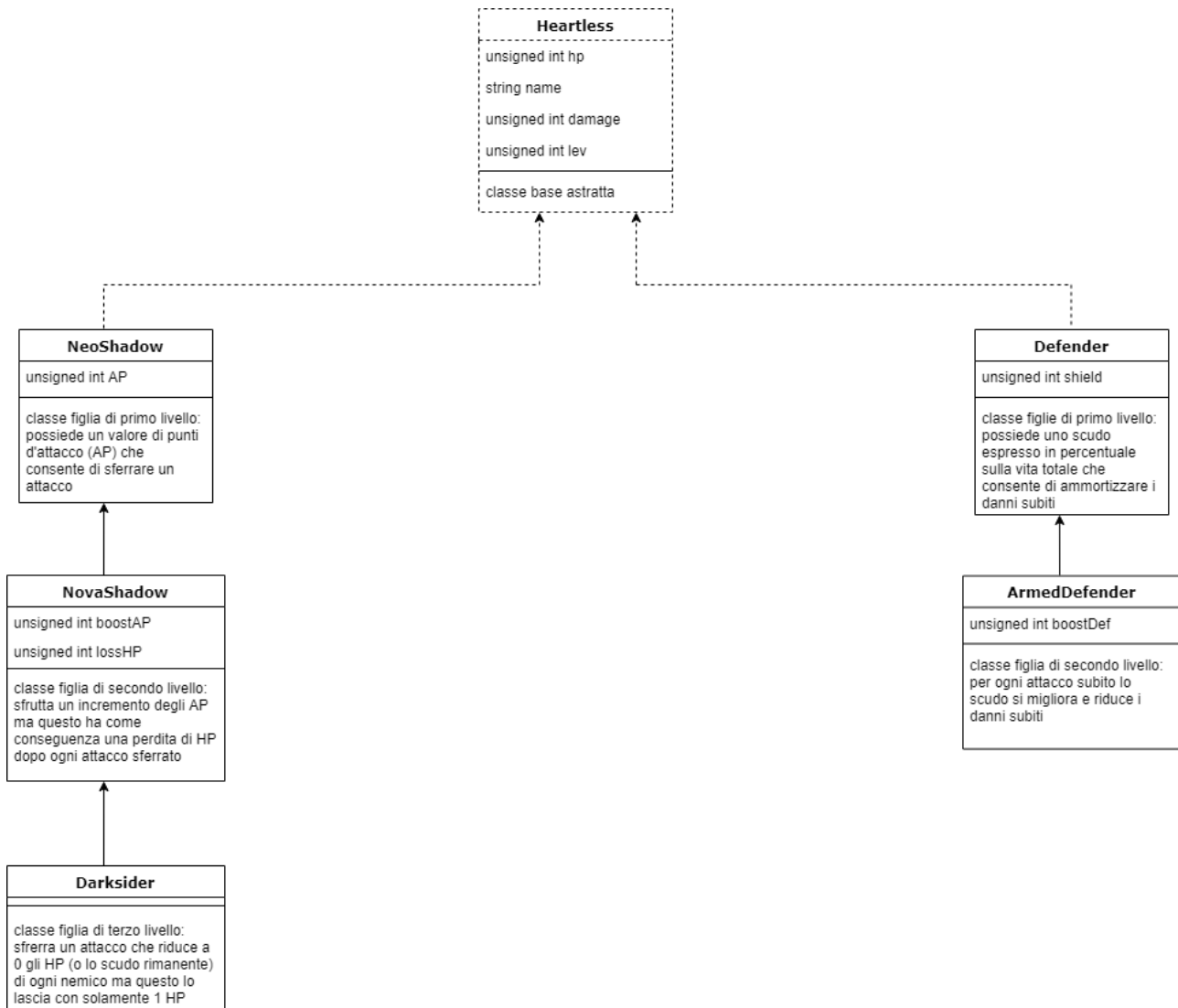
È stata una scelta personale quella di non introdurre una schermata di modifica dei personaggi. Per dimostrare la comunque funzionante possibilità di alterare i dati memorizzati, si è scelto di sfruttare unicamente la schermata di battaglia tra i diversi personaggi contenuti nel Qontainer.

Compilazione

Per effettuare la compilazione è necessario il file .pro incluso nella cartella del progetto. Questo perché vengono utilizzate alcune funzionalità di c++11; per questo motivo, in compilazione saranno necessari i comandi qmake e make. Nella directory del progetto viene inoltre fornito un file .xml con una collezione di Heartless da poter caricare nel programma e utilizzare come test. Nella sottocartella GUI (contenente le classi utilizzate per l'interfaccia grafica) è presente anche un file .qss utilizzato come foglio di stile per la gestione dell'aspetto grafico.

Gerarchia

La gerarchia del progetto si sviluppa su quattro livelli:



La classe base Heartless è astratta, presenta infatti metodi virtuali e non consente l'allocazione di istanze. Essa include le informazioni riguardanti il nome del personaggio, la salute e il danno subito fino a quel momento (che serve a determinare la salute rimanente) e infine il livello del personaggio, che viene calcolato in base alle caratteristiche di ogni sotto-personaggio.

Il primo livello di gerarchia presenta due classi figlie della principale, NeoShadow e Defender. La prima rappresenta un Heartless con la possibilità di sferrare un attacco. Ogni NeoShadow presenta un valore di AP (attack point) che determina l'intensità del colpo sferrato. I NeoShadow però non possono difendersi dagli attacchi subiti. La seconda classe figlia Defender è invece dotata di un attributo "scudo" che consente loro di incassare i colpi subiti, decrementando il valore dello scudo (espresso in percentuale sul valore totale della salute) prima di subire una perdita della vita rimanente. Essi però non possono sferrare attacchi.

Passando al secondo livello, otteniamo un'ulteriore classe figlia per ognuna delle due classi del livello sovrastante. NovaShadow è la classe figlia di NeoShadow. Ogni personaggio di questa

classe può effettuare un attacco con un valore di AP incrementato ma questo comporta una perdita di HP ad ogni attacco effettuato. ArmedDefender è invece figlia di Defender; ogni personaggio assorbe gli attacchi grazie allo scudo e, ad ogni attacco incassato, lo scudo aumenta la propria capacità di assorbimento degli urti, riducendo il danno subito.

L'ultimo livello include invece solamente una classe, figlia di NovaShadow. La classe Darksider definisce un personaggio in grado di scagliare un colpo che elimina completamente il personaggio avversario (se esso non presenta alcuno scudo) o riduce a zero il valore dello scudo dello stesso; questo attacco speciale ha però un costo ed infatti lascia il personaggio "attaccante" con un solo punto vita.

Polimorfismo

La classe base Heartless include i seguenti metodi virtuali (puri e non):

- `virtual ~Heartless () = default;` distruttore di default per la classe astratta.
- `virtual string print () const;` restituisce una stringa riportante i dettagli dell'oggetto.
- `virtual string getType () const = 0;` restituisce una stringa riportante il genere dell'oggetto, cioè la sottoclasse a cui esso appartiene.
- `virtual Heartless* clone () const = 0;` restituisce un puntatore ad una copia dell'oggetto implicito sul quale viene effettuata l'invocazione del metodo.
- `virtual unsigned int getAP () const;` restituisce il valore dei punti attacco dell'oggetto.
- `virtual unsigned int getShield () const;` restituisce il valore percentuale dello scudo dell'oggetto.
- `virtual unsigned int attack () = 0;` effettua un attacco con l'oggetto che ha invocato il metodo e di conseguenza sfrutta le sue caratteristiche particolari.
- `virtual unsigned int defend (unsigned int) = 0;` simula un attacco contro l'oggetto che ha invocato il metodo, reagendo in maniera coerente con la specie dell'oggetto di invocazione.
- `virtual void xml (QXmlStreamWriter&) const;` permette il salvataggio dell'oggetto di invocazione su file xml.

Per quanto riguarda i metodi `getAP()` e `getShield()`, la scelta di utilizzarli come metodi virtuali è dovuta puramente alla comodità nella gestione delle istanze. Al contrario, i rimanenti metodi sono necessariamente da definire come virtuali in quanto sono presenti in tutte le sottoclassi ma con comportamenti differenti in base alla tipologia dell'oggetto. Essi vengono quindi implementati in ogni figlio della classe base, mentre i due citati sopra vengono invece sovrascritti.

Container

Il progetto contiene un container template ed una classe di puntatori "smart" `DeepPtr`, come richiesto dalla consegna. Il contenitore è stato realizzato con una DDL (Doubly Linked List) in modo da favorire le rimozioni in punti casuali della lista, ritenendola quindi la struttura più adatta al progetto. A causa della struttura di Qt, spesso vengono utilizzati indici anziché iteratori nello scorrimento delle istanze. Per questo motivo, l'operatore `+` gioca un ruolo fondamentale.

Il contenitore `Container<T>` conterrà quindi oggetti di tipo `DeepPtr`, che a loro volta conterranno puntatori "smart" di oggetti di tipo `Heartless`.

Xml

Come metodo di salvataggio e caricamento dei dati, è stato scelto l'utilizzo di file xml. Tramite l'apposito menù è possibile selezionare il file da cui importare i dati oppure la destinazione per il salvataggio degli stessi. Per motivi ovvi, la selezione potrà essere effettuata solamente tra file con estensione .xml. I metodi di salvataggio e caricamento vengono implementati in una classe esterna denominata *MyXml.cpp* e che viene sfruttata come se fosse una nuova libreria. Essa contiene infatti i due metodi pubblici **save** e **load**. Di seguito viene riportato un esempio di salvataggio di un personaggio (altri sono disponibili nel file *qheartless.xml* in allegato al progetto).

```
<Heartless>
  <NeoShadow>
    <Nome>Betrayner</Nome>
    <HP>45</HP>
    <Danno>45</Danno>
    <AP>3</AP>
  </NeoShadow>
</Heartless>
```

Per gestire in maniera più sicura i costruttori delle varie classi e non consentire troppa libertà di modifica ad ipotetici utenti finali, si è deciso di memorizzare solamente i valori degli attributi principali delle classi figlie di primo livello e trascurare gli incrementi speciali delle classi di livello più basso. Per questo motivo, gli attributi **boostAP** e **lossHP** per **NovaShadow** e **boostDef** per **ArmedDefender** non vengono memorizzati e sono quindi riportati al valore base iniziale dopo il caricamento delle istanze.

Dal punto di vista puramente descrittivo possiamo riassumere come segue: gli incrementi valgono solamente mentre la battaglia è in corso (applicazione in esecuzione); al termine essi ripartono da zero (applicazione chiusa e riaperta oppure caricamento di dati salvati).

GUI

La parte grafica dell'applicazione è divisa in più file per gestire separatamente le varie schermate di interazione:

- **MyMainWindow.cpp** contiene e gestisce tutte le sotto-parti della schermata grafica; possiede quindi metodi in grado di gestire gli elementi da visualizzare a video e permette il collegamento tra la parte grafica e il modello logico di gestione dei dati.
- **Index.cpp** contiene la home page; possiede metodi per interagire con i tre pulsanti visualizzati all'avvio. Non interagisce con il modello logico del container.
- **Insert.cpp** contiene il form dedicato all'inserimento di un nuovo personaggio all'interno del container. In base alla specie di personaggio selezionato, viene visualizzata un'immagine che lo raffigura.
- **RemoveL.cpp** contiene la schermata che consente di cercare e successivamente rimuovere un oggetto dalla lista. La ricerca può essere effettuata per genere o per nome.
- **Listino.cpp** contiene una tabella raffigurante tutti i personaggi attualmente memorizzati e ne consente la rimozione.

- **Battle.cpp** permette di visualizzare la schermata per simulare una battaglia tra due personaggi distinti selezionati dal listino memorizzato.

In ogni pagina è inoltre presente un menù a barra per la navigazione rapida tra le varie schermate ed il salvataggio/caricamento dei dati. Sempre all'interno della directory GUI, il file **MyXml.cpp** svolge i compiti di salvataggio e caricamento già descritti in precedenza, mentre il file **Model.cpp** è il responsabile dell'interazione tra il Container (con i propri puntatori ed il puntatore profondo **DeepPtr**) e le classi di personaggi derivate da Heartless.

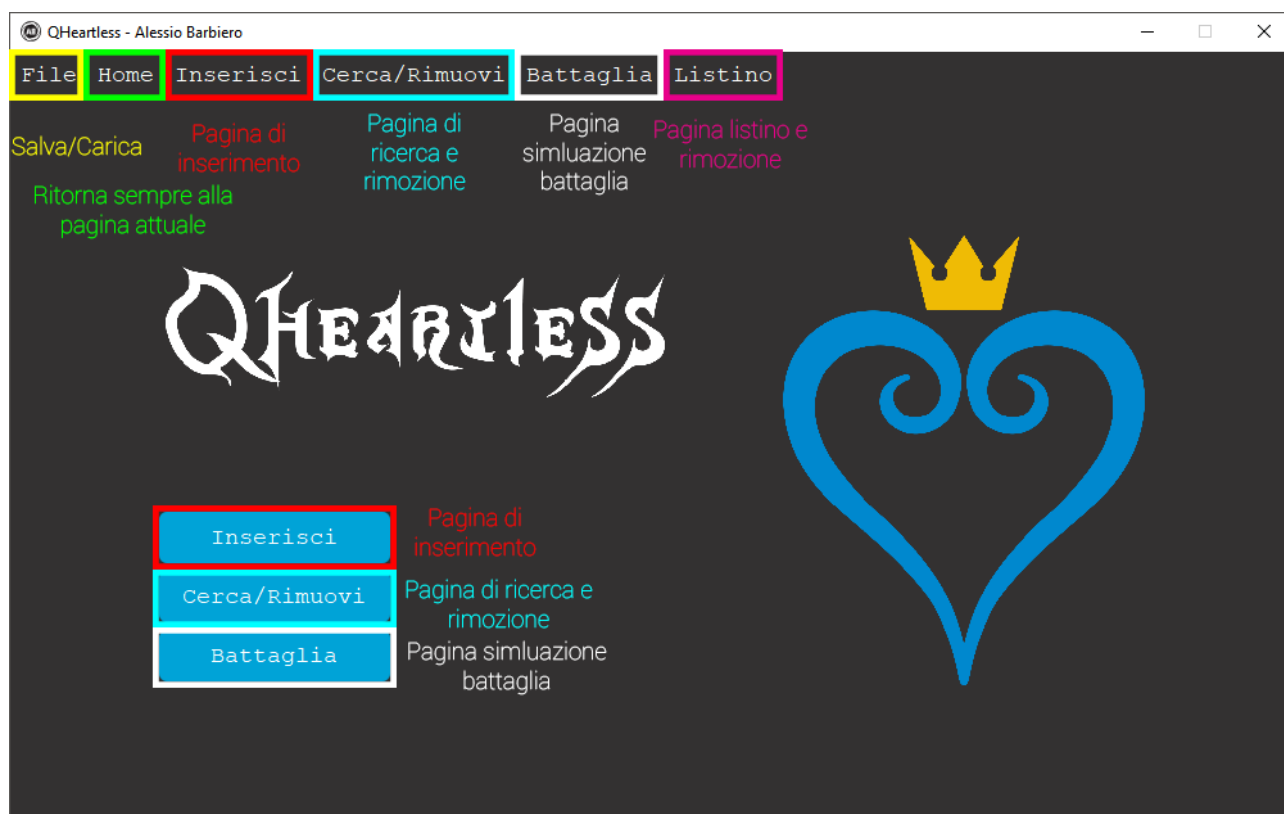
Viene infine fornito un file **style.qss** da utilizzare come foglio di stile per la gestione dell'aspetto delle componenti grafiche.

Manuale d'utilizzo

Il layout dell'applicazione è stato pensato per essere semplice ed intuitivo; di seguito vengono riportati gli screenshots delle schermate con qualche consiglio d'utilizzo.

Alcune annotazioni:

- I report evidenziati in rosso sono i personaggi che hanno raggiunto OHP e che quindi vengono considerati sconfitti. Non possono né sferrare né subire attacchi.
- Nel form di inserimento sono stati implementati metodi per verificare che i dati inseriti siano corretti. In caso contrario, una messagebox informa dell'errore.
- Nella schermata di battaglia, una messagebox di errore compare se viene selezionato lo stesso personaggio sia per attaccare che per difendere o se uno dei due non viene selezionato.
- In entrambe le schermate di rimozione, compare una messagebox di errore se non viene selezionato alcun elemento dalla tabella.



Homepage

QHeartless - Alessio Barbiero

File Home Inserisci Cerca/Rimuovi Battaglia Listino

Form per l'inserimento dei dati

Aggiunta nuovo Heartless

Genere: **NeoShadow**

Nome:

HP:

AP:




Immagine del personaggio scelto

Annulla inserimento

Azzera form

Aggiungi

Conferma inserimento

Inserimento

QHeartless - Alessio Barbiero

File Home Inserisci Cerca/Rimuovi Battaglia Listino

Visualizza tutti gli elementi o seleziona quello da eliminare

Genere	Nome	Livello	HP totali	HP rimasti	AP	% Scudo
NeoShadow	Betrayer	12	45	0	3	0
NovaShadow	Barneswood	178	530	530	36	0
Defender	Grrdy	156	780	780	0	66
ArmedDefen...	Boothstomp...	178	890	890	0	100
Darksider	Grangke	4196	9999	9999	99	0
Defender	Porpuff	249	1245	1245	0	99
ArmedDefen...	Grayiverse	600	3000	3000	0	57
NeoShadow	Tustink	153	700	700	13	0
NovaShadow	Arisullivan	376	1350	1350	53	0
Darksider	Drackelly	3186	7538	7538	86	0
ArmedDefen...	Greeneswim...	738	3694	3694	0	1
ArmedDefen...	Sibim	1893	9465	9465	0	97
NeoShadow	Mephistoph...	216	917	917	33	0
Defender	Grahaminiar	463	2316	2316	0	43
NovaShadow	Wecoffin	581	2396	2396	51	0

Elimina

Elimina l'elemento selezionato

Listino e eliminazione

QHeartless - Alessio Barbiero

File Home Inserisci Cerca/Rimuovi Battaglia Listino

Criterio di ricerca Termine da ricercare

Genere Cerca

Genere	Nome	Livello	HP totali	HP rimasti	AP	% Scudo
NeoShadow	Betrayner	12	45	0	3	0
NovaShadow	Barneswood	178	530	530	36	0
Defender	Grrdy	156	780	780	0	66
ArmedDefen...	Boothstomp...	178	890	890	0	100
Darksider	Grangke	4196	9999	9999	99	0
Defender	Porpuff	249	1245	1245	0	99
ArmedDefen...	Grayiverse	600	3000	3000	0	57
NeoShadow	Tustink	153	700	700	13	0
NovaShadow	Arisullivan	376	1350	1350	53	0
Darksider	Drackelly	3186	7538	7538	86	0
ArmedDefen...	Greeneswim...	738	3694	3694	0	1
ArmedDefen...	Sibim	1893	9465	9465	0	97
NeoShadow	Mephistoph...	216	917	917	33	0
Defender	Grahaminiar	463	2316	2316	0	93

Elimina

Elimina l'elemento selezionato

Visualizza l'esito della ricerca o
seleziona elemento da eliminare

Ricerca e eliminazione

QHeartless - Alessio Barbiero

File Home Inserisci Cerca/Rimuovi Battaglia Listino

Scegli l'attaccante: Seleziona dalla tabella il personaggio che attacca

Genere	Nome	Livello	HP totali	HP rimasti	AP	% Scudo
NeoShadow	Betrayner	12	45	0	3	0
NovaShadow	Barneswood	178	530	530	36	0
Defender	Grrdy	156	780	780	0	66
ArmedDefend...	Boothstomper	178	890	890	0	100
Darksider	Grangke	4196	9999	9999	99	0
Defender	Porpuff	249	1245	1245	0	99

Scegli il difensore: Seleziona dalla tabella il personaggio che difende

Genere	Nome	Livello	HP totali	HP rimasti	AP	% Scudo
NeoShadow	Betrayner	12	45	0	3	0
NovaShadow	Barneswood	178	530	530	36	0
Defender	Grrdy	156	780	780	0	66
ArmedDefend...	Boothstomper	178	890	890	0	100
Darksider	Grangke	4196	9999	9999	99	0
Defender	Porpuff	249	1245	1245	0	99

ATTACCA!

Simula attacco

Simulazione battaglia

Tempistiche

Analisi dei requisiti e progettazione: 3h

Studio individuale delle librerie Qt: 4h

Realizzazione gerarchie + debug: 8h + 2h

Realizzazione contenitore + debug: 9h + 2h

Realizzazione classe puntatori DeepPtr + debug: 4h + 1h

Progettazione e realizzazione GUI tramite Qt: 12h

Test e debug GUI e interazione con modello: 5h

Realizzazione file .qss + ricerca e modifica immagini: 2h

Stesura relazione: 4h

Tempo totale realizzazione progetto: 56h

N.B. Il tempo elevato di debug della parte grafica è dovuto ad un problema di interazione riscontrato dell'utilizzo di `QList<T>` per la memorizzazione degli indici di tabella; l'errore era semplice da risolvere ma ha richiesto diverso tempo per essere individuato.