

JPokeBattle

Leonardo Ganzaroli 1961846

Febbraio 2025

Indice

1	Descrizione generale	2
2	Progetto sviluppato	4
2.1	GUI	4
2.1.1	Menù principale	4
2.1.2	Leaderboard	5
2.1.3	Schermata di gioco	6
2.1.4	Game Over	7
2.1.5	Cambio mosse	8
2.2	Struttura	8
2.2.1	Pokémon	9
2.2.2	Statistiche	9
2.2.3	Mosse	9
2.3	Design Pattern	10
2.3.1	Strategy	10
2.3.2	Singleton	11
2.4	Paradigmi di programmazione	12
2.4.1	Stream	12

Elenco delle figure

1	Menù	4
2	Leaderboard con alcuni record	5
3	Gioco appena avviato	6
4	3 mosse	6
5	No record	7
6	Nuovo record	7
7	Cambio mossa	8
8	Strategy pattern	10
9	Singleton pattern	11

1 Descrizione generale

Il progetto da sviluppare può essere riassunto come:

"Replicare il gameplay delle lotte all'interno dei videogiochi della serie Pokémon, facendo riferimento alla prima generazione".

Per una descrizione più specifica si può far riferimento alla seguente lista di funzionalità richieste per questo progetto.

(Quelle implementate sono segnate con un ✓, quelle implementate parzialmente invece con ✓)

Specifiche richieste:

- Feature minime
 - Implementare i 3 starter, le loro statistiche e le loro mosse base: ✓
 - * Bulbasaur → Ruggito e Azione
 - * Charmander → Ruggito e Graffio
 - * Squirtle → Colpocoda e Azione
 - Creare una GUI utilizzando Java Swing o JavaFX. ✓
 - Implementare le schermate classiche presenti in un videogioco: ✓
 - * Menù principale
 - * Schermata di combattimento
 - * Schermata di "Game Over"
 - * ...
 - Far affrontare al giocatore una serie di lotte fino alla sua sconfitta. ✓
- Feature tipiche
 - Implementare tutte le mosse dei Pokémon scelti ignorando però i cambiamenti di stato. ✓

- Preservare lo stato dei pokémon del giocatore nella serie di lotte. ✓
- Implementare una schermata leaderboard che mantenga i 10 record migliori. ✓
- Feature extra
 - Implementare punti individuali e punti allenamento che migliorino le capacità dei pokémon sulla base delle vittorie. ✓
 - Implementare i meccanismi di passaggio di livello ed evoluzione dei Pokémon, incluso l'apprendimento di nuove mosse. ✓
 - Implementare animazioni, riproduzione di audio clip, ed altri effetti grafici. ✓
 - Implementare unit test del backend.
 - Creare un Jar eseguibile.
 - Implementare strategie per un comportamento “intelligente” degli avversari.
 - Implementare una modalità di gioco testuale via terminale, selezionabile in alternativa alla GUI.

Elementi implementati in modo parziale (in ordine):

- Le mosse che portano solamente a dei cambiamenti di stato, che applicano degli effetti continui o comunque “particolari” sono state ignorate.
- I punti allenamento non sono stati implementati.
- Il sistema dei livelli è presente ma non quello delle evoluzioni.
- L’unica “animazione” presente è solamente un delay tra la scritta “Appare pokémon X”, la comparsa dello sprite e la riproduzione del verso. Non sono presenti altri effetti grafici.

2 Progetto sviluppato

Questa sezione contiene una breve descrizione della struttura del progetto e dell'interfaccia grafica.

2.1 GUI

L'interfaccia grafica si può suddividere in 5 parti, ognuna creata ad hoc:

2.1.1 Menù principale

La finestra che contiene il menù principale.

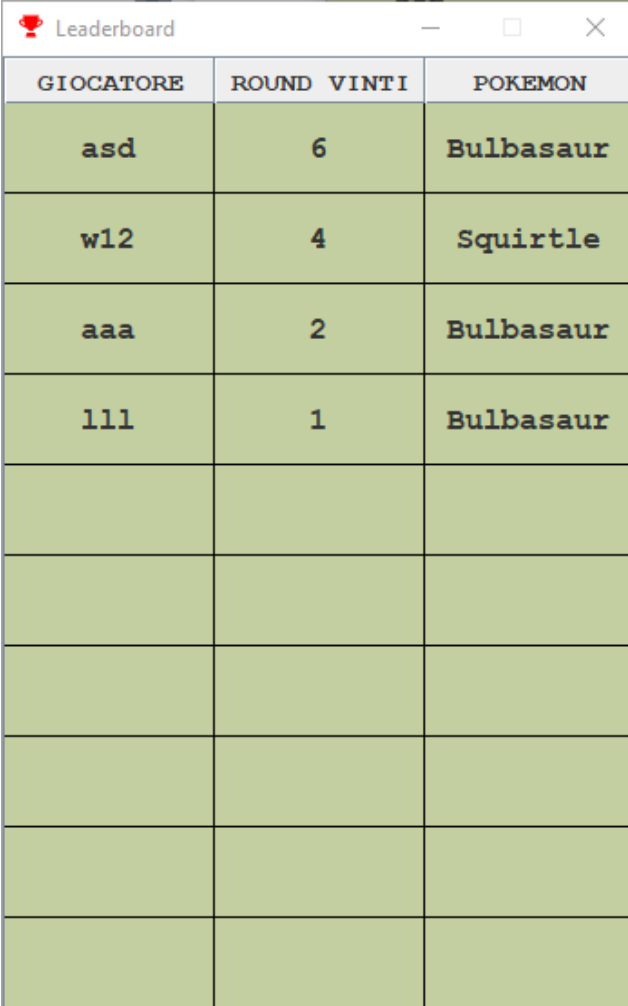


Figura 1: Menù

Ai lati sono presenti 2 bottoni che permettono di aprire la Leaderboard (SX) e di giocare (DX). In basso è presente una combo box che permette di scegliere il pokémon da utilizzare.

2.1.2 Leaderboard

Contiene i migliori 10 record in termini di round vinti.



A screenshot of a window titled "Leaderboard" with a red trophy icon. The window contains a table with three columns: "GIOCATORE", "ROUND VINTI", and "POKEMON". The table lists the top 10 records, with the first four rows containing data and the remaining six rows being empty. The data rows show players "asd", "w12", "aaa", and "111" with 6, 4, 2, and 1 wins respectively, all using "Bulbasaur".

GIOCATORE	ROUND VINTI	POKEMON
asd	6	Bulbasaur
w12	4	Squirtle
aaa	2	Bulbasaur
111	1	Bulbasaur

Figura 2: Leaderboard con alcuni record

2.1.3 Schermata di gioco

L'effettiva schermata di gioco, creata basandosi su quella dei vari giochi.

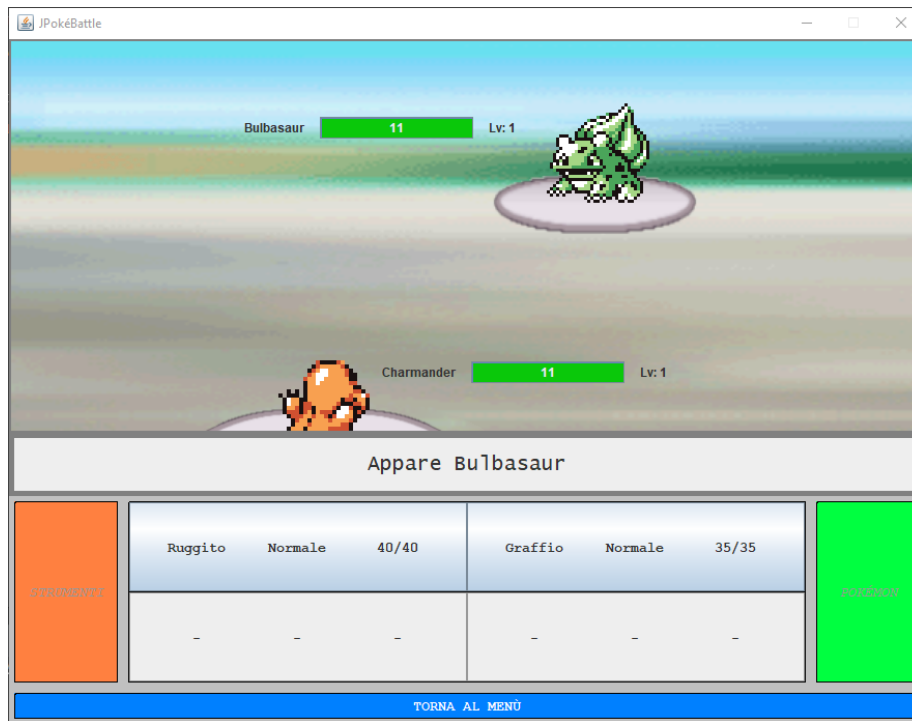


Figura 3: Gioco appena avviato

La parte inferiore della schermata andrà a variare con l'apprendimento di nuove mosse da parte del pokémon.



Figura 4: 3 mosse

2.1.4 Game Over

Appare dopo la sconfitta del giocatore, nel caso in cui venga stabilito un nuovo record viene data la possibilità di salvarlo.

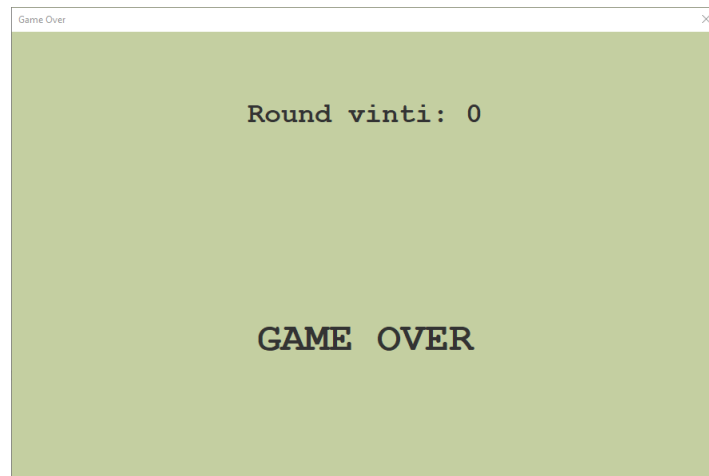


Figura 5: No record

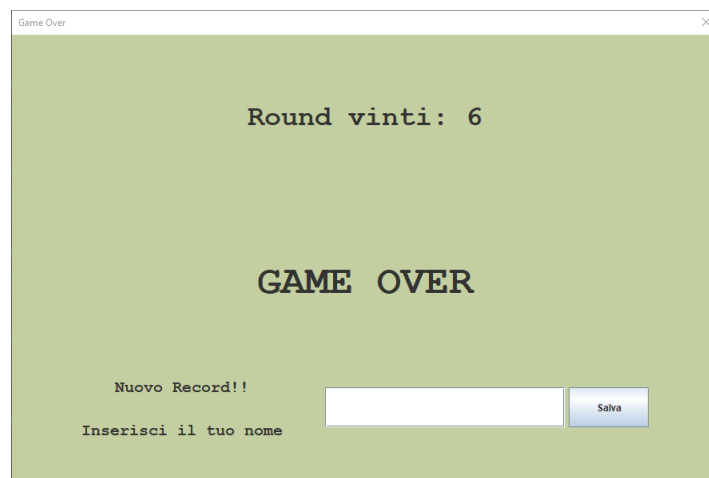


Figura 6: Nuovo record

2.1.5 Cambio mosse

Usata quando il pokémon conosce già 4 mosse e vuole impararne una nuova, permette di scegliere se e quale mossa rimuovere.

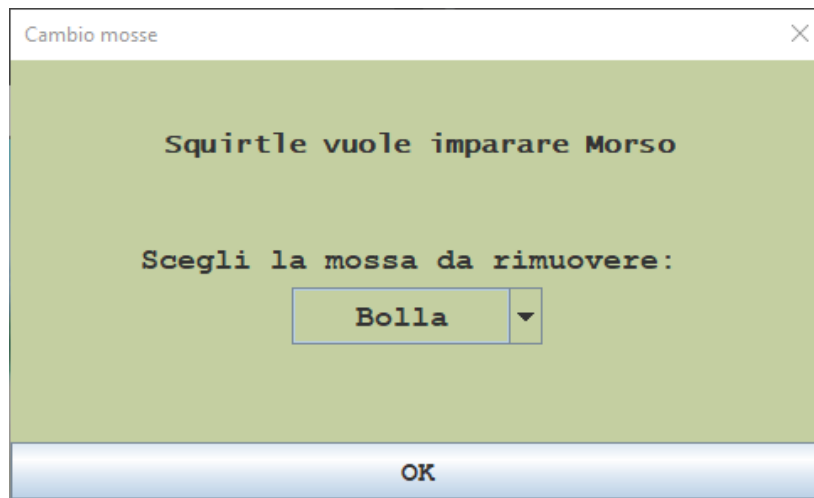


Figura 7: Cambio mossa

In questo caso si sta rimuovendo Bolla per far imparare Morso.

2.2 Struttura

Per avere maggiore flessibilità viene fatto uso di file testuali contenenti tutte le informazioni riguardanti i pokémon e le loro mosse. In questo modo viene data la possibilità di creare/modificare i valori a piacimento rendendo possibile all'utente l'uso di elementi custom.

Per quanto riguarda le classi utilizzate e la loro struttura si identificano 3 elementi principali.

2.2.1 Pokémon

La classe che rappresenta un pokémon, contiene le statistiche e le mosse in aggiunta a varie funzioni. Sono state inoltre create 2 sottoclassi per rappresentare il giocatore ed i nemici, ognuna con funzioni specifiche necessarie per il funzionamento del gioco.

2.2.2 Statistiche

La classe che raggruppa tutti i valori associati ad un certo pokémon: Vita, Attacco, Difesa,

2.2.3 Mosse

Considerando i requisiti del progetto è stato possibile dividere ogni singola mossa in una di queste 5 categorie:

1. Danneggia l'avversario

- (a) Danni normali
- (b) Danni speciali

La formula usata per calcolare i danni è identica nei due casi ma i valori usati sono leggermente diversi.

Formula della prima generazione:

$$\left(\frac{\left(\frac{2 \times Livello \times Critico}{5} + 2 \right) \times Potenza \times \left(\frac{A}{D} \right)}{50} + 2 \right) \times STAB \times Tipo1 \times Tipo2 \times Random$$

2. Cambia le statistiche

- (a) Dell'avversario
- (b) Proprie

3. Mosse con effetti particolari

L'ultimo tipo è stato inserito per permettere di creare eventuali mosse che non rientrano nelle altre categorie, in questo modo c'è la possibilità di aggiungere quelle non implementate in futuro (vedere sezione successiva).

2.3 Design Pattern

In questo progetto sono presenti 2 pattern.

2.3.1 Strategy

Per implementare le mosse è stato utilizzato il pattern Strategy, in questo modo si potrà scegliere la funzione più adatta da associare ad una certa mossa. Permette inoltre di creare ulteriori funzioni in caso di necessità.

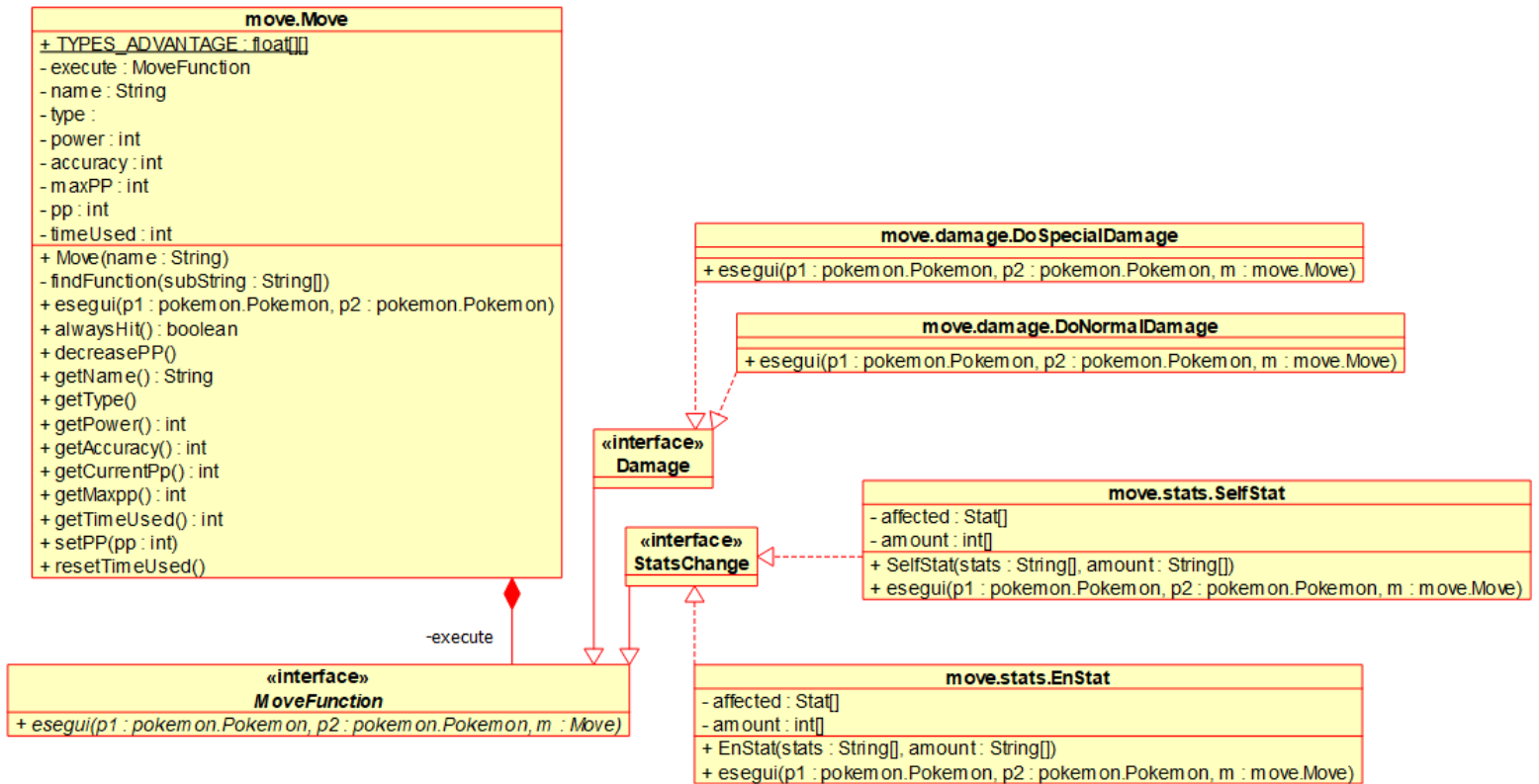


Figura 8: Strategy pattern

2.3.2 Singleton

Dato che la leaderboard è una finestra a sè stante apribile dal menù viene utilizzato il pattern Singleton per evitare l'apertura di finestre multiple.

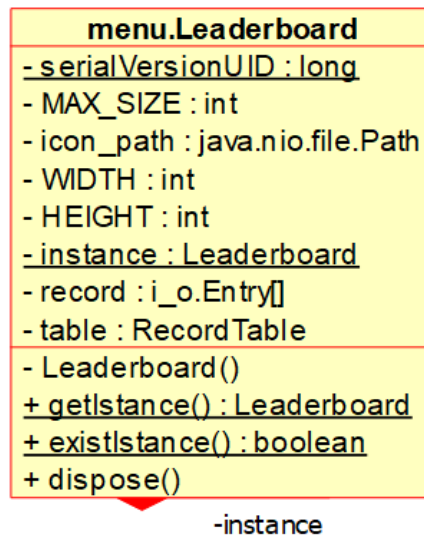


Figura 9: Singleton pattern

2.4 Paradigmi di programmazione

2.4.1 Stream

Sono presenti 2 funzioni nel codice in cui si fa uso degli stream, entrambe hanno a che fare con la lettura di file testuali:

(N.B. `br` è il `BufferedReader`)

1. Nella funzione *allPokemon* della classe *FileRw*, dove è presente la seguente riga:

Codice 1 Crea un insieme contenente tutti i nomi dei pokémon presenti nel file.

```
result = br.lines()
        .filter(x→x.contains("name : "))
        .map(x→x.replace("name : ", ""))
        .collect(Collectors.toCollection(TreeSet :: new));
```

2. Nel ciclo della funzione *fill* della classe *Entry*:

Codice 2 Legge i record dal file e crea un array di oggetti *Entry* che verranno poi utilizzati nel ciclo.

```
for (Entry e : br.lines()
        .filter(x→!x.isEmpty())
        .map(x→new Entry(x.split(" ")))
        .sorted(Comparator.comparing(Entry :: getStreak).reversed())
        .limit(MAX_RECORD)
        .toArray(Entry[] :: new)) do
    ...
end for
```
