Duel Simulator

Simulatore di duelli fantasy

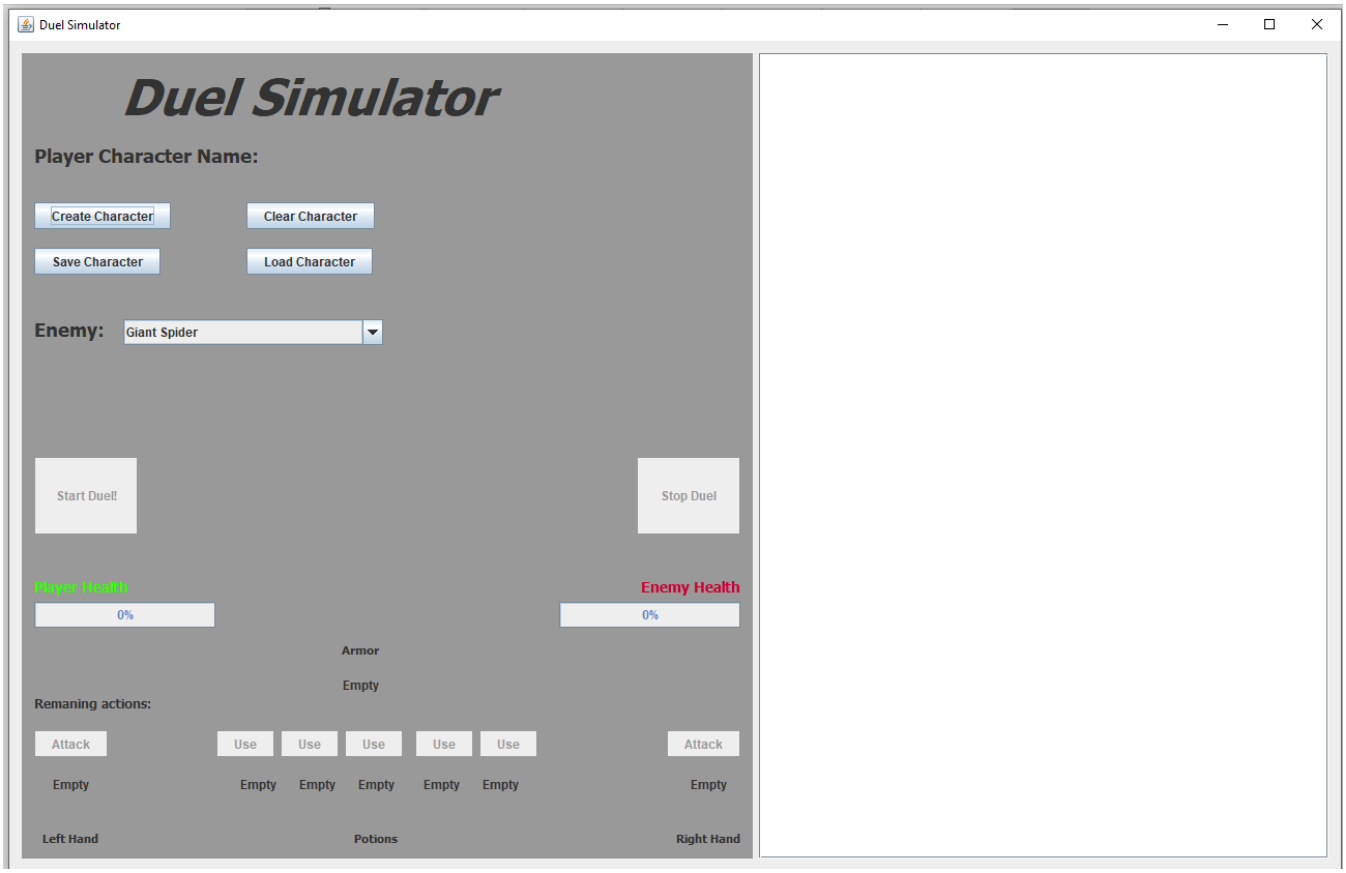
Luongo Danilo 3FIT Progetto informatica

Duel Simulator è un simulatore di duelli fantasy in cui l'utente può crearsi un **personaggio** o utilizzarne uno già creato in passato e farlo **combattere** con un **nemico** scelto. Il personaggio ha a disposizione un'**armatura,** una o 2 **armi** e fino a 5 **pozioni**. Nella schermata è presente una **dashboard** che ricorda gli ultimi eventi del **duello**.

L'**utente** all'avvio del programma ha l'opzione di creare un personaggio, che lo porta ad un altra **finestra** dove seleziona le caratteristiche del nuovo personaggio, di caricarne uno già esistente (file **.chr**), di deselezionare il personaggio creato/caricato, di salvare il personaggio selezionato e di selezionare, attraverso una **combobox**, il nemico da affrontare.
Quando un personaggio e un nemico sono stati scelti, un pulsante **"Inizia duello!"** sarà disponibile e se cliccato i pulsanti di preparazione duello e quello appena usato non saranno più disponibili, mentre tutti gli altri diventeranno disponibili. Tra quest'ultimi c'è il pulsante "**Ferma duello"** che se cliccato farà il contrario di "Inizia duello!".
Durante il duello il personaggio ha a disposizione un numero di **azioni** pari alla sua **velocità**. Le sue azioni possono essere usate per:

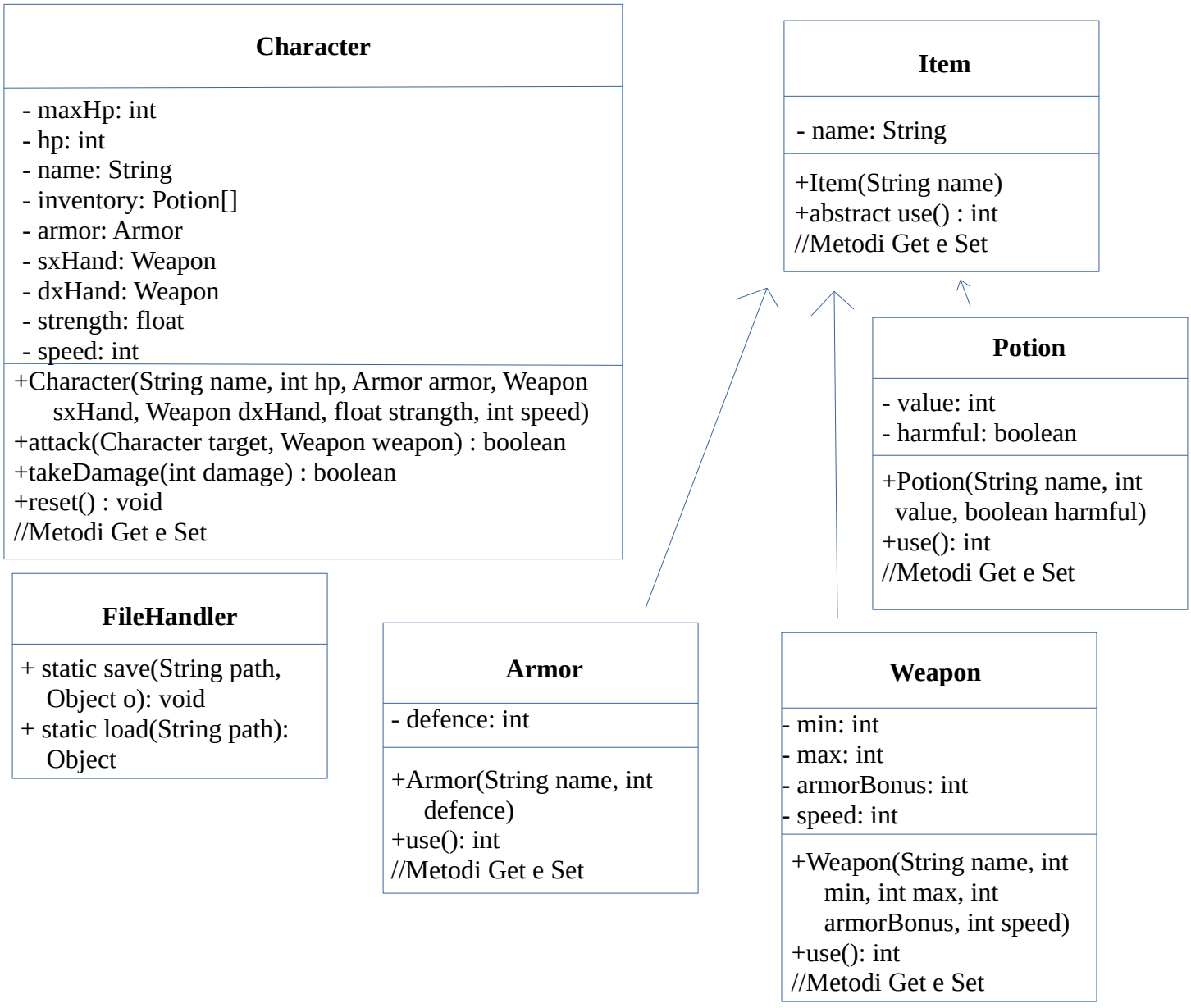
- **attaccare** il nemico con una delle sue armi (se preme il pulsante "Attacca" sopra la scritta "Mano sinistra" attacca con l'arma nella mano sinistra, altrimenti con quella nella mano destra);
- **usare** una delle 5 pozioni attraverso i relativi pulsanti (come mostrato nella figura in basso).

Il duello può finire in 3 modi: o il personaggio **vince**, o **perde** o l'utente **ferma** il duello con il pulsante omonimo.



Informazioni tecniche

Grafico classi



Il **progetto** è costituito da **6** classi (**Character**, **Item**, **Armor**, **Potion**, **Weapon**, **FileHandler**) e da 1 form per l'**interfaccia grafica**. Mi sono servito inoltre di una classe esterna **CustomOutputStream** (da www.codejava.net) per trasferire l'output della console alla dashboard. Grazie ai pulsanti "Save Character" e "Load Character" di quest'ultima è possibile **salvare** e **caricare** un oggetto della classe Character attraverso la classe **FileHandler**, che ha 2 metodi statici: uno per salvare (save(String path, Object o)) e uno per caricare (load(Object o)).

La classe **Character** (Personaggio) ha come attributi:

- int hp: la vita del personaggio;
- int maxHp: la vita totale del personaggio;
- String name: il nome del personaggio;
- Potion[] inventory: l'inventario del personaggio, rappresentato da un array di Potion, classe discussa più avanti;
- Armor armor: l'armatura del personaggio, rappresentata da un oggetto di classe Armor, classe discussa più avanti;
- Weapon sxHand: l'arma presente nella mano sinistra del personaggio, rappresentata da un oggetto di classe Weapon, classe discussa più avanti;
- Weapon dxHand: l'arma presente nella mano destra del personaggio, rappresentata da un oggetto di classe Weapon, classe discussa più avanti;
- float strength: la forza del personaggio, rappresentato da un moltiplicatore che verrà applicato al danno delle armi;
- int speed: la velocità del giocatore, rappresentata da un numero di azioni che il personaggio può effettuare in un turno.

Tra i metodi non-costruttori e ignorando quelli di tipo Get e Set troviamo:

- boolean attack(Character target, Weapon weapon): ricava i danni dall'arma data come parametro e li infligge (attraverso il prossimo metodo descritto) al target dato come parametro. Se il target muore con questo attacco restituisce true, altrimenti false;
- boolean takeDamage(int damage): sottrae i danni ricevuti come parametro all'attributo hp. Se il personaggio muore (cioè hp <= 0) restituisce true, altrimenti false;
- void reset(): reimposta gli attributi ai loro valori di default.

La classe **Item** (Oggetto) è una classe astratta e ha un attributo, String name, che rappresenta il nome dell'oggetto. Ignorando il metodo costruttore e quelli di tipo Get e Set è presente solo il metodo int use(), un metodo astratto che dovrà essere definito da ogni sottoclasse di Item attraverso l'uso dell'**override.** La classe Item ha **3 sottoclassi**, Armor, Potion e Weapon.

La classe **Armor** (Armatura), sottoclasse di Item, ha un solo attributo, int defence, che rappresenta quanto difende dai danni. Ignorando il metodo costruttore e quelli di tipo Get e Set è presente solo il metodo int use(), sovracritto (tramite override) a quello della classe madre, e ritorna 0, poiché un'armatura ha una funzione passiva.

La classe **Potion** (Pozione) ha due attributi:

- int value: il valore numerico del suo uso (esempio: una pozione della cura recupera value punti vita);
- boolean harmful: se la pozione è dannosa, questa variabile ha come valore true, altrimenti false.

Ignorando il metodo costruttore e quelli di tipo Get e Set è presente solo il metodo int use(), sovracritto (tramite override) a quello della classe madre, e restituisce value se harmful è false, altrimenti restituisce -value (cioè value negato).

La classe **Weapon** (Arma) ha 4 attributi:

- int min: è il minimo danno che può recare l’arma;
- int max: è il massimo danno che può recare l’arma;
- int armorBonus: è il bonus da aggiungere all’armatura;
- int speed: è il numero di azioni richieste dal personaggio per attaccare con quest’arma.

Ignorando il metodo costruttore e quelli di tipo Get e Set è presente solo il metodo int use(), sovracritto (tramite override) a quello della classe madre, e restituisce un numero casuale tra gli attributi min e max.