Questa avventura testuale, sviluppata interamente in linguaggio Java, offre un’interfaccia utente con la quale interagire e supporta il salvataggio e caricamento della partita.

**TextAdventure**

La struttura del gioco è basata sulla creazione e gestione di una mappa composta da varie stanze, definite genericamente come Room e specializzate in base alle interazioni e agli eventi di gioco. Infatti, le stanze, gestite come classi, conterranno gli oggetti specifici per la locazione che esse rappresentano assieme ai metodi che, a loro volta, rappresenteranno le interazioni possibili all’interno della data stanza.

Per poter compiere azioni nel gioco, l’utente è invitato a inserire il comando e l’eventuale oggetto col quale si vuole interagire nel JTextField apposito. In seguito alla pressione del tasto Invio o del JButton “Submit”, il comando è processato in modo da essere riconosciuto al fine di attivare l’interazione o l’evento di gioco desiderato.

Gli **oggetti di gioco** saranno definiti come istanze della classe Stobj o di classi che ereditano da essa. Ad esempio, le armi apparterranno alla classe Weapon, le pozioni alla classe Potion e così via per tutti gli oggetti che necessitano di metodi o argomenti specifici al fine dell’interazione con essi.

**Comandi e Parser**I comandi accettati dal sistema dovranno essere coniugati alla seconda persona singolare.   
A seguito della digitazione, la frase di comando inserita dall’utente per compiere l’azione verrà processata dalla classe Parser:

* Al fine di essere identificato, il primo comando inserito verrà confrontato con una lista di comandi preimpostati;
* Se il confronto va a buon fine, verrà ricercata una corrispondenza tra l’eventuale oggetto specificato nella frase di comando e tutti gli oggetti presenti nella stanza;
* Se l’oggetto specificato non è presente nella stanza, ne verrà ricercata la corrispondenza all’interno dell’inventario del giocatore;
* Se con l’oggetto inserito ci si riferisce ad una “porta” e se il numero di porte all’interno della stanza è > 1, il parser sarà pronto a effettuare un controllo per capire a quale porta ci si riferisce, basandosi sulla direzione specificata;
* Infine, il parser restituirà un oggetto ParserOutput che conterrà i comandi e l’oggetto di gioco (più l’eventuale direzione) necessari a successive computazioni per soddisfare la richiesta dell’utente.

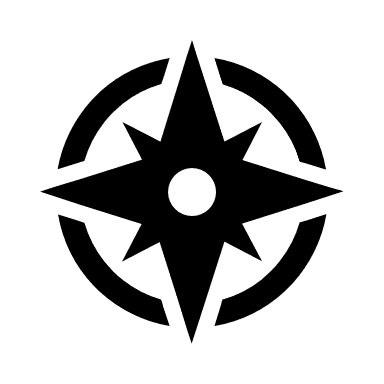
**Interfaccia utente**Il package gameInterface definisce due classi, UI e VisiblityManager, che insieme gestiscono l’interfaccia utente mediante l’utilizzo di JavaSWING.  
Nella classe UI, vengono definiti tutti i componenti che costituiscono l’interfaccia: alla base c’è una window alla quale vengono aggiunti vari panel a seconda di cosa descrivono. Questi contengono i JButtons, i JLabels, le JTextAreas e il JTextField che l’utente utilizzerà per inserire i comandi.  
La classe VisibilityManger, invece, non definisce componenti ma utilizza quelli istanziati in UI per definire le schermate di gioco: menu principale, fase di combattimento, schermate di caricamento fallito o di conferma dell’uscita dal gioco, o più banalmente anche lo stesso testo che viene visualizzato all’interno della JTextArea principale.  
Il funzionamento delle due classi è subordinato alla classe Game che comunica con UI tramite un Listener, quindi aspettando un’azione dell’utente come un click o un invio e rispondendo di conseguenza, e utilizza i metodi di VisibilityManager per creare le “transizioni” dell’interfaccia.

**Player**La dotazione del giocatore è specificata all’interno dell’istanza della classe Player. In essa sono definiti:

* Punti vita (HP), correnti e totali;
* Le monete possedute;
* L’inventario, come lista di Stobj, contenente gli oggetti raccolti e posseduti;
* Il valore “dmg” corrispondente al danno che il giocatore può infliggere al nemico e derivato dall’arma posseduta;
* Il valore “def” corrispondente alla difesa, derivato dall’armatura posseduta, che corrisponde alla riduzione del danno ricevuto da parte dei nemici;

**Mappa di gioco**Cosi come accennato prima, la mappa è gestita come un oggetto che conterrà, a sua volta, oggetti rappresentanti le varie stanze di gioco.  
Le stanze di gioco sono definite genericamente a partire dalla classe Room.   
Per le stanze in cui avvengono particolari eventi o interazioni, e che contengono oggetti unici o prevedono la presenza di nemici, vengono utilizzate istanze di classi, contenute nel package roomSet, le quali ereditano per estensione o variazione funzionale tramite Override, la struttura principale da Room.   
Per muoversi da una stanza all’altra, il giocatore dovrà specificare la direzione rappresentata dai quattro punti cardinali principali.  
In caso di presenza di elementi che non consentono l’avanzamento nella data direzione come porte chiuse, nemici o indovinelli, per proseguire il giocatore dovrà prima liberare la strada (aprendo la porta, sconfiggendo il nemico o risolvendo l’indovinello).   
Gli oggetti di gioco saranno contenuti in una lista propria di ogni stanza ed i nemici sono definiti come istanze di classi che ereditano da SuperMonster.

Shaman’s  
secret  
Room



N

E

O

Shaman’s  
Room

S

Storage

Mount  
Entrance

Wyvern  
Lair

Mount  
base

Forest  
ending

Cliff

Nymph’s  
Lake

Path to  
Lake

Forest  
Clearing

North Gate

BlackSmith

AlchemistShop

Village main square

House

**Salvataggio e Caricamento**Le funzioni di salvataggio e caricamento sono gestite interamente dal package fileManager che, tramite la classe fileHandler, si occupa di:

* Serializzare gli oggetti map e player al fine di trasformarli in una sequenza di byte per poi memorizzarli su due file separati al momento del salvataggio;
* Recuperare i dati dei rispettivi oggetti map e player al momento del caricamento per poi restituirli al chiamante;

**Gioco**Il nucleo dell’applicazione risiede nel package gameCore che contiene:

* La classe di definizione della lista dei comandi che il sistema sarà in grado di accettare come validi (CommandList);
* La classe di definizione del giocatore (Player);
* La classe di definizione della mappa di gioco (Map);
* La classe Story contenente il metodo che sarà in grado di attivare l’evento di gioco a cui il giocatore punta tramite l’inserimento del comando desiderato. La frase di comando che questa classe ottiene viene prima processata dal parser al fine di ricondurre il comando inserito e l’oggetto specificato a quelli che il sistema è in grado di riconoscere.   
  Gli eventuali messaggi di gioco generati a seguito dell’evento verranno visualizzati su interfaccia tramite le funzioni della classe VisibilityManager;
* La classe Game contenente il main, da cui il gioco viene lanciato, come classe principale del progetto, istanzia tutti gli elementi necessari alla esecuzione del software, e predispone le impostazioni di gioco di default nel caso di nuova partita o le prende dai file di salvataggio in caso di caricamento della stessa.  
  Al suo interno è definita una inner class, choiceHandler, che si occupa della gestione degli eventi riguardanti l’interazione dell’utente con l’interfaccia (pressione dei vari pulsanti e inserimento del testo) e smista i compiti alle classi di competenza al momento del richiamo della funzione necessaria a risolvere tale evento   
  (ad esempio: ottiene l’input dell’utente e lo passa a Story dopo averlo fatto processare dal parser;   
  richiama le funzioni di salvataggio e caricamento dal fileHandler su richiesta dell’utente;   
  modifica la schermata di gioco all’attivazione della fase di combattimento, eccetera).

**Combattimento**Se in una stanza dovesse esserci un nemico, il giocatore è costretto a sconfiggerlo per poter proseguire. A tal fine, nel momento in cui l’utente digita il comando “attacca” o un eventuale suo sinonimo, l’interfaccia viene temporaneamente modificata per visualizzare la fase di combattimento.   
Quest’ultima viene gestita tramite la pressione di pulsanti dedicati alle varie azioni permesse in combattimento:

* Attacca: i danni sono calcolati tramite una funzione random. I danni provocati dal nemico variano da un minimo di 1 ad un massimo definito nella sua istanziazione (a cui va sottratto il punteggio definito dall’attributo “defence” dell’armatura posseduta); i danni provocati dal giocatore invece, si basano sull’attributo “damage” dell’arma posseduta e variano da un minimo di damage-5 ad un massimo pari a damage stesso.  
  Oltre al punteggio della vita e dell’attacco, all’istanziazione del nemico vengono definiti gli attributi “hitRate” e “dodgeRate” che, rispettivamente, indicano la percentuale di successo del suo attacco e la percentuale di possibilità dell’evitare il danno provocato dal giocatore.
* Usa Pozione: così come è possibile utilizzare una pozione al di fuori del combattimento tramite comando scritto, il giocatore può premere questo pulsante per ripristinare la sua salute. Nel caso la salute fosse già al massimo o se non si possiedono pozioni, verrà visualizzato un messaggio per notificarlo.
* Scappa: consente al giocatore di ritirarsi dal combattimento, ritornando alla stanza precedente.
* Osserva: consente al giocatore di avere informazioni relative alle condizioni di salute del nemico che si sta affrontando.

Se a seguito di un attacco da parte del nemico, la salute del giocatore dovesse divenire <= 0, verrà visualizzata la schermata di sconfitta e sarà data la possibilità al giocatore di tornare al menu principale.

**Conclusioni**Il programma è stato ideato e scritto il più genericamente possibile in modo da poter essere facilmente ampliabile o modificabile:  
infatti modificando il contenuto della mappa e delle stanze che la compongono e con piccoli aggiustamenti nella classe Story, è possibile realizzare avventure testuali completamente diverse, pur basandosi sulla stessa struttura del gioco.

A causa dell’emergenza COVID-19, lavorare a distanza è stato leggermente più complicato del previsto.   
Ci siamo organizzati tramite l’utilizzo di piattaforme come Discord e Telegram, per la comunicazione e il pair programming, e github per gli aggiornamenti e le modifiche al codice.