RPG Adventure Game - Demo

Overview

RPG Adventure Game è un gioco di ruolo testuale in stile "play by chat" che richiama i classici giochi testuali del passato. Questa versione rappresenta una demo funzionale e una base di partenza per un progetto più ampio che sarà sviluppato in Unity con C#.

Il progetto nasce dalla passione per i videogiochi e dalla volontà di creare delle fondamenta solide per un futuro RPG completo. Le meccaniche di base sono ispirate al concetto del Tamagochi, adattate a un contesto fantasy con guerrieri e maghi. L'attenzione è stata posta principalmente sulle meccaniche di gioco e sull'architettura software piuttosto che sull'aspetto estetico, trattandosi di un progetto dimostrativo.

Funzionalità del Gioco

Creazione e Gestione Personaggi

- Creazione di personaggi di due tipologie: **Guerriero** e **Mago**
- Sistema di statistiche (vita, stamina, danno, denaro, livello)
- Gestione della mana per i maghi
- Sistema di allenamento per migliorare le statistiche
- Salvataggio e caricamento dei personaggi

Sistema di Combattimento

- Combattimento a turni contro mostri
- Creazione dinamica di mostri (Goblin, Troll)
- Sistema di drop oggetti casuali (implementazione base da completare)
- Recupero della stamina che sarà esteso con un sistema overtime durante i momenti di riposo e non azione

Sistema di Inventario

- Inventario dinamico per ogni personaggio
- Gestione di armi, armature, pozioni e oggetti vari
- Sistema di equipaggiamento con bonus statistici
- Ordinamento degli oggetti per nome, tipo o valore

Esplorazione Dungeon

- Dungeon configurabili con il Builder Pattern

- **Goblin Cave** e **Swamp of Trolls** disponibili
- Ricompense in oro per i dungeon completati
- Sistema di esplorazione con incontri casuali

Interfaccia

- Menu testuali a scelta multipla strutturati gerarchicamente
- Navigazione intuitiva tra menu principali e sottomenu
- Feedback continuo sullo stato del personaggio
- Sistema di logging per tracciare le azioni

Tecnologie e Design Pattern Implementati

Design Pattern

- **Factory Pattern**: Creazione di personaggi e mostri
- **Composite Pattern**: Sistema di menu gerarchico
- **Iterator Pattern**: Iterazione su inventari e oggetti
- **Builder Pattern**: Costruzione flessibile dei dungeon
- **Strategy Pattern**: Algoritmi di ordinamento dell'inventario
- **Observer Pattern**: Sistema di notifiche per recupero stamina
- **Singleton Pattern**: Gestione centralizzata del logging

Tecnologie Core Java

- **Collections Framework**: Gestione di liste e mappe tipizzate
- **Generics**: Type safety in tutto il progetto
- **Java I/O**: Persistenza dei personaggi su file
- **Logging**: Sistema di log completo con java.util.logging
- **JUnit Testing**: Test di integrazione e unit test

Tecnologie Avanzate

- **Stream API & Lambda**: Operazioni funzionali sui dati
- **Reflection**: Accesso dinamico a campi privati
- **Mockito**: Test isolati con mock objects

Sicurezza

- **Input Sanitization**: Validazione e pulizia degli input utente
- **Exception Shielding**: Gestione sicura delle eccezioni

- **Controlled Error Propagation**: Prevenzione di leak di informazioni

Setup e Esecuzione

Prerequisiti

- Java 11 o superiore
- Maven per la gestione delle dipendenze
- JUnit e Mockito per i test

Compilazione ed Esecuzione

```bash
# Compilazione
mvn compile

# Esecuzione
mvn exec:java

# Esecuzione test
mvn test
```

File di Configurazione

Il gioco crea automaticamente una cartella `saves/` per i salvataggi dei personaggi. I file vengono salvati in formato Properties con estensione `.save`.

Architettura del Progetto

Struttura dei Package

```
rpg/
├── factory/           # Creazione personaggi
├── factoryMonster/    # Creazione mostri
├── composite/         # Sistema menu
├── iterator/          # Inventario e oggetti
├── builder/           # Costruzione dungeon
├── strategy/          # Algoritmi ordinamento
├── observer/          # Sistema notifiche
├── menu/              # Menu di gioco
├── combat/            # Sistema combattimento
├── rpgIO/             # Gestione I/O
├── rpgSecurity/       # Validazione e sicurezza
└── logger/            # Sistema logging
```

Decisioni di Design

- **Modularità**: Ogni componente è isolato con responsabilità specifiche
- **Estensibilità**: Facile aggiunta di nuovi tipi di personaggi, mostri e dungeon
- **Sicurezza**: Input validation e exception handling robusti
- **Testabilità**: Architettura che facilita unit testing e mocking

Giustificazioni Tecniche

Factory Pattern

Scelto per la creazione di personaggi e mostri per garantire:

- **Incapsulamento**: La logica di creazione è centralizzata
- **Estensibilità**: Facile aggiunta di nuovi tipi senza modificare il codice esistente
- **Validation**: Controllo centralizzato degli input di creazione

Composite Pattern

Implementato per il sistema di menu per ottenere:

- **Uniformità**: Trattamento uniforme di menu e sottomenu
- **Flessibilità**: Struttura gerarchica facilmente modificabile
- **Semplicità**: Client code che non distingue tra leaf e composite

Iterator Pattern

Utilizzato per l'inventario per fornire:

- **Astrazione**: Accesso agli oggetti senza esporre la struttura interna
- **Sicurezza**: Iterazione controllata e type-safe
- **Uniformità**: Interfaccia standard per l'iterazione

Builder Pattern

Applicato ai dungeon per permettere:

- **Configurabilità**: Creazione flessibile di dungeon con parametri variabili
- **Leggibilità**: Codice di creazione più comprensibile
- **Validazione**: Controllo della correttezza durante la costruzione

Strategy Pattern

Implementato per l'ordinamento dell'inventario per garantire:

- **Flessibilità**: Algoritmi di ordinamento intercambiabili

- **Estensibilità**: Facile aggiunta di nuovi criteri di ordinamento
- **Single Responsibility**: Ogni strategia ha una responsabilità specifica

Observer Pattern

Implementato per osservare i cambiamenti di stamina del giocatore per ottenere:

- **Disaccoppiamento**: UI e logica di gioco separate
- **Reattività**: Notifiche automatiche quando la stamina cambia
- **Estensibilità**: In futuro potrà essere esteso per implementare il recupero automatico della stamina con lo scorrere del tempo reale

Limitazioni Attuali

- **Interfaccia**: Solo testuale, nessuna GUI
- **Contenuti**: Limitato a 2 classi di personaggi e 2 tipi di dungeon
- **Gestione Inventario**: Mancata gestione ottimale degli oggetti dell'inventario
- **Sistema di Combattimento**: Richiede miglioramenti e maggiore complessità
- **Grafica**: Assente, focus sulle meccaniche
- **Persistenza**: File system locale, nessun database

Sviluppi Futuri

Questo progetto rappresenta le fondamenta per un RPG più complesso che sarà sviluppato in Unity con C#. Gli aspetti che verranno espansi includono:

- **Interfaccia Grafica**: Migrazione da testo a GUI completa
- **Contenuti**: Più classi, dungeon, oggetti e meccaniche
- **Sistema di Quest**: Missioni strutturate
- **Economy**: Sistema economico più complesso
- **Environment Diversi**: Possibilità di spostarsi in zone diverse con bonus e malus specifici basati sulla zona e sull'equipaggiamento
- **Gestione Inventario**: Miglioramento del sistema di inventario con funzionalità avanzate

Test Coverage

Il progetto include test completi:

- **Integration Test**: Verifica dell'interazione tra componenti
- **Unit Test con Mockito**: Test isolati di singoli componenti
- **Test di Sicurezza**: Validazione dell'input sanitization

La suite di test copre i componenti critici del sistema garantendo la stabilità delle funzionalità core.

Questo progetto dimostra l'applicazione pratica di design pattern e tecnologie Java in un contesto di game development, fornendo una base solida per progetti futuri più ambiziosi.