**Documentazione delle classi**

**Classi di logica**

**Zelda**

Classe applicazione. Istanzia l’interfaccia, carica le impostazioni e istanzia di conseguenza gli oggetti di modello che costituiscono l’applicazione oltre ad inizializzare le componenti per il log e le interazioni con il database.

Gestisce gli eventi quali la pressione del pulsante di login, l’utilizzo del campo username e la pressione dei tasti di gioco, in collaborazione con le altre classi.

**InterfaceBuilder**

Classe dedicata alla costruzione e stilizzazione dell’interfaccia grafica

**ServerLogXml**

Classe main del server di log. Riceve dal client di gioco gli eventi per il log in formato xml, li valida e li aggiunge al file di log.

**GameModel**

Classe che costituisce il modello del gioco, riceve i comandi da eseguire da Zelda (ovvero dal giocatore) e li applica tramite la classe GameCharacter. Mantiene lo stato della partita in termini di informazioni riguardo il posizionamento di nemici e personaggio, e di informazioni riguardo l’andamento della stessa, come il punteggio e le vite del giocatore. Gli elementi grafici di gioco invece vengono acceduti tramite GameView.

**GameCharacter**

Classe che definisce le informazioni e le possibilità di movimento e di attacco del personaggio controllato dal giocatore.

**GameEnemy**

Classe che definisce le informazioni e le possibilità di movimento e di attacco di un nemico, così come una elementare intelligenza artificiale per poter scegliere le migliori opzioni di movimento per raggiungere il giocatore e attaccare quando possibile.

**GameTile**

Classe che gestisce le informazioni riguardo un elemento della matrice costituente il campo di gioco.

**TurnHandler**

Tread dedicato alla gestione dei turni, e al passaggio da un nemico all’altro per la rispettiva azione automatica.

**GameView**

Classe che gestisce le componenti visuali del gioco. Istanzia e posiziona gli elementi grafici del gioco, fornisce metodi per accedervi o indicarne posizioni specifiche. Fornisce metodi per avviare le animazioni, e aggiorna lo stato delle stesse, per indicare se sono terminate o meno, per il controllo del proseguimento del gioco.

**GraficTile**

Elemento grafico che rappresenta una casella della matrice costituente il campo di gioco visibile.

Espone metodi per l’aggiornamento grafico in base alle informazioni date dal rispettivo GameTile contenuto nel GameModel.

**DBManager**

Classe contenente i metodi per accedere al database, in particolare per le operazioni di inserimento, lettura ed eliminazione dei Record presenti nello schema Ranking.

**Classi statiche**

**LocalCacheOperations**

Classe statica che contiene metodi per salvare e recuperare la cache locale.

Accede alle classi applicative per salvare e ripristinare lo stato della partita.

**EventLoggerXML**

Classe statica che contiene i metodi per creare un oggetto EventLogXML a partire dalla descrizione dell’evento, serializzarlo e inviarlo via socket al server di log.

Valida il messaggio XML tramite XSD prima dell’invio, inviando solo i messaggi che superano la validazione.

**ReaderSettingsXML**

Classe statica che contiene i metodi per leggere il file di impostazione xml, validarne il contenuto e ottenere l’oggetto SettingsXML da utilizzare nell’applicazione. Nel caso la validazione del contenuto fallisce, vengono restituite impostazioni di default mentre il file xml viene sovrascritto con queste.

**GameUtils**

Classe statica contenente costanti e metodi di utilità per le classi di logica dedicate alla gestione della partita.

**Strutture dati**

**Record**

Struttura dati dedicata all’archiviazione di un Record conseguito dal giocatore corrente.

Viene utilizzato come modello per l’estrazione o registrazione di dati presenti nel Database

**EventLogXML**

Classe che contiene le informazioni di log da inviare via socket al server di log.

Richiede come parametro la descrizione dell’evento, mentre ricava le altre informazioni (indirizzo IP e timestamp) alla creazione.

Viene creato, serializzato e inviato via socket dalla classe EventLoggerXML.

**CacheData, GameCacheData, ElementCacheData**

Strutture dati serializzabili utilizzate da LocalCacheOperations per salvare e caricare lo stato dell’applicazione via file binario.

**SettingsXML, KeyAssociation, ServerAddress**

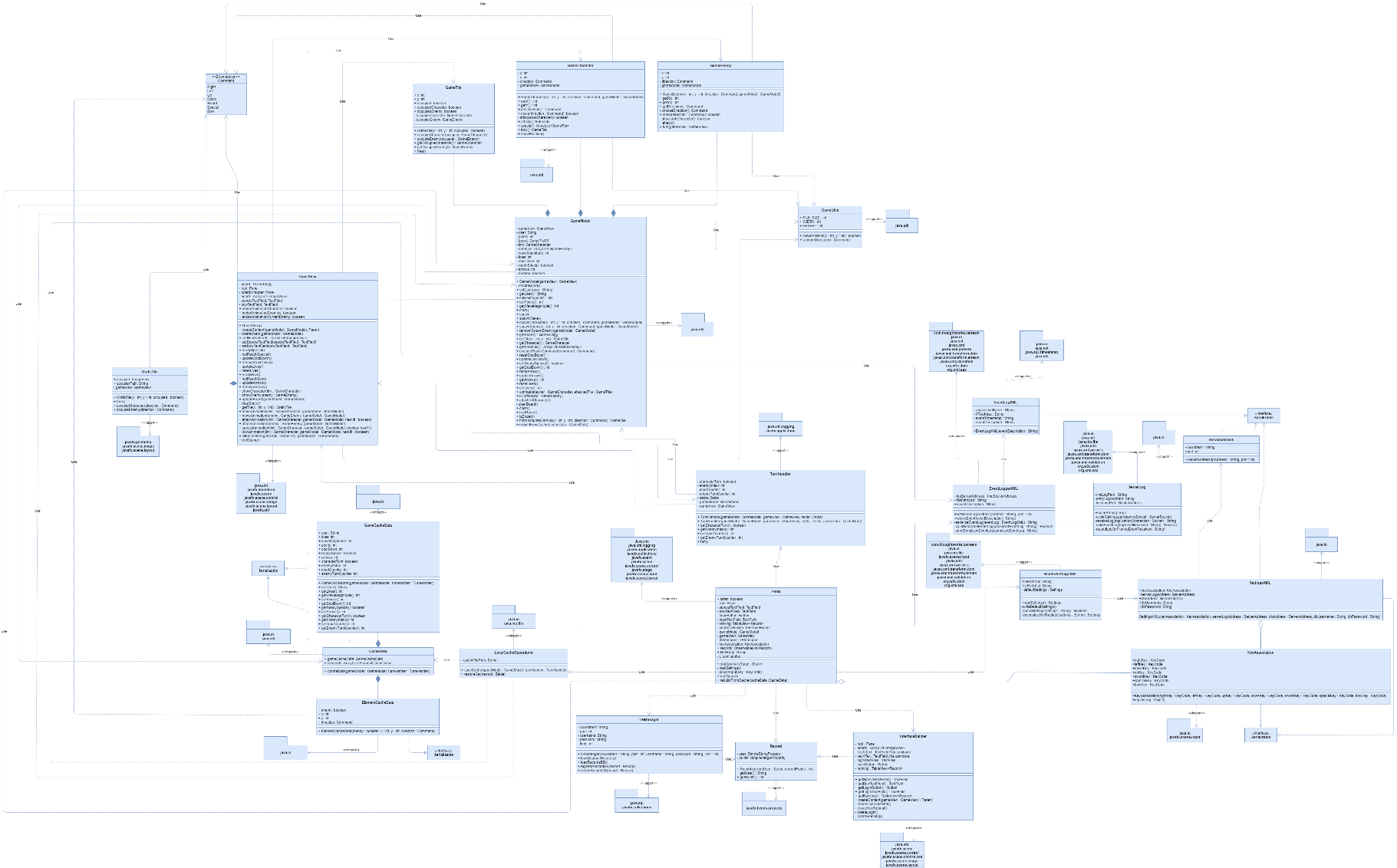
Strutture dati utilizzate per la deserializzazione delle impostazioni xml, e per serializzazione in caso di scrittura delle impostazioni default.

All’avvio Zelda ottiene l’oggetto SettingsXML da ReaderSettingsXML, e con questo oggetto vengono configurati i vari componenti dell’applicazione.

Enumerazioni

**Command**

Enumerazione che indica il comando di gioco inserito dall’utente.

**Schema UML**