Relazione per "Progettazione e Sviluppo del Software"

Cristian Postovan
Tommaso Conti

29 dicembre 2025

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

PuzzleRogue è un videogioco single-player che fonde la logica classica del **Sudoku** con meccaniche di progressione e gestione risorse tipiche del genere **Roguelike**.

Il giocatore veste i panni di un personaggio che deve affrontare una "Run" (spedizione), composta da una sequenza di livelli (dungeon) di difficoltà crescente. Ogni livello è rappresentato da una griglia di Sudoku da completare. Il completamento dei livelli permette di avanzare nella spedizione e accumulare punteggio.

Il gioco introduce elementi strategici che vanno oltre la semplice risoluzione del puzzle:

- **Sistema di Vite:** Il giocatore ha un numero limitato di vite per l'intera Run. Ogni errore commesso nel Sudoku riduce le vite. Esaurite le vite, la partita termina (Game Over).
- **Buff Permanenti:** Tra una partita e l'altra, il giocatore può spendere i punti accumulati per acquistare potenziamenti permanenti (Buff) che faciliteranno le spedizioni future (es. vite extra, protezione dagli errori, capacità inventario aumentata).
- **Inventario e Oggetti:** Durante la partita, il giocatore può gestire un inventario di oggetti consumabili che forniscono aiuti immediati (es. rivelare una cella).

L'obiettivo finale è completare tutti i livelli previsti (fino al Boss finale) o massimizzare il punteggio prima della sconfitta.

Requisiti Funzionali

- **Gestione Utente:**
  - Creare un nuovo profilo utente tramite nickname.
  - Mantenere la persistenza dei dati utente (punteggio totale, statistiche, buff sbloccati) tra diverse sessioni di gioco.
  - Acquistare e potenziare Buff permanenti utilizzando i punti accumulati.
- **Gestione della Partita (Run):**
  - Avviare una nuova partita selezionando un personaggio.
  - Ogni partita deve utilizzare una "fotografia" (snapshot) dei Buff posseduti dall'utente al momento dell'avvio; modifiche successive ai Buff non devono influenzare la partita in corso.
  - Gestire una sequenza di 10 livelli di difficoltà crescente.
  - Gestire il sistema di vite: decrementare le vite in caso di errore (salvo protezioni attive).
  - Decretare la vittoria della Run al completamento dell'ultimo livello o la sconfitta all'esaurimento delle vite.
- **Gameplay (Livello):**
  - Generare proceduralmente griglie di Sudoku valide e con soluzione unica.
  - Permettere l'inserimento di numeri e note nelle celle.
  - Validare le mosse dell'utente in tempo reale.
  - Gestire l'uso di oggetti consumabili dall'inventario per ottenere aiuti.
- **Sistema di Punteggio:**
  - Calcolare il punteggio al termine di ogni livello basandosi su: difficoltà, bonus vari, e moltiplicatori attivi.
  - Fornire un resoconto dettagliato (Breakdown) del punteggio al termine della partita.

Requisiti Non Funzionali

- **Interfaccia Utente:** L'interfaccia deve essere responsiva e fornire feedback visivi immediati per azioni corrette, errate o uso di oggetti.
- **Persistenza Locale:** Tutti i dati (progressi, salvataggi, profili) devono essere salvati localmente (SQLite), senza dipendenze da server remoti.
- **Performance:** La generazione dei puzzle e la validazione delle mosse devono avvenire in tempo reale senza latenze percepibili.
- **Portabilità:** Il software deve essere eseguibile su sistemi desktop standard con supporto Java.

1.2 Modello del Dominio

Il **Giocatore** (User) intraprende delle **Spedizioni** (Run) all'interno del gioco. Ogni Spedizione rappresenta una sessione di gioco distinta ed è costituita da una sequenza progressiva di **Livelli**. Il cuore di ciascun Livello è la risoluzione di una **Griglia Sudoku** (SudokuGrid), che rappresenta la sfida logica che il giocatore deve superare per proseguire.

Il Giocatore è un'entità persistente che mantiene uno storico dei progressi e accumula risorse (punti) tra una spedizione e l'altra. Utilizzando queste risorse, il Giocatore può sbloccare e migliorare dei **Potenziamenti** (Buff). Tali potenziamenti sono elementi chiave che influenzano le regole delle future Spedizioni, fornendo vantaggi passivi come vite extra, celle già rivelate o capacità di inventario aumentata.

Durante una Spedizione, il giocatore può acquisire e conservare nel proprio inventario degli **Oggetti** (Item). Questi sono risorse consumabili che forniscono aiuti immediati per superare situazioni di stallo o correggere errori durante la risoluzione della Griglia Sudoku. La Spedizione termina quando il giocatore esaurisce le vite a disposizione o completa con successo tutti i livelli previsti.

Gli elementi costitutivi il problema e le loro relazioni sono sintetizzati in Figura 1.1.

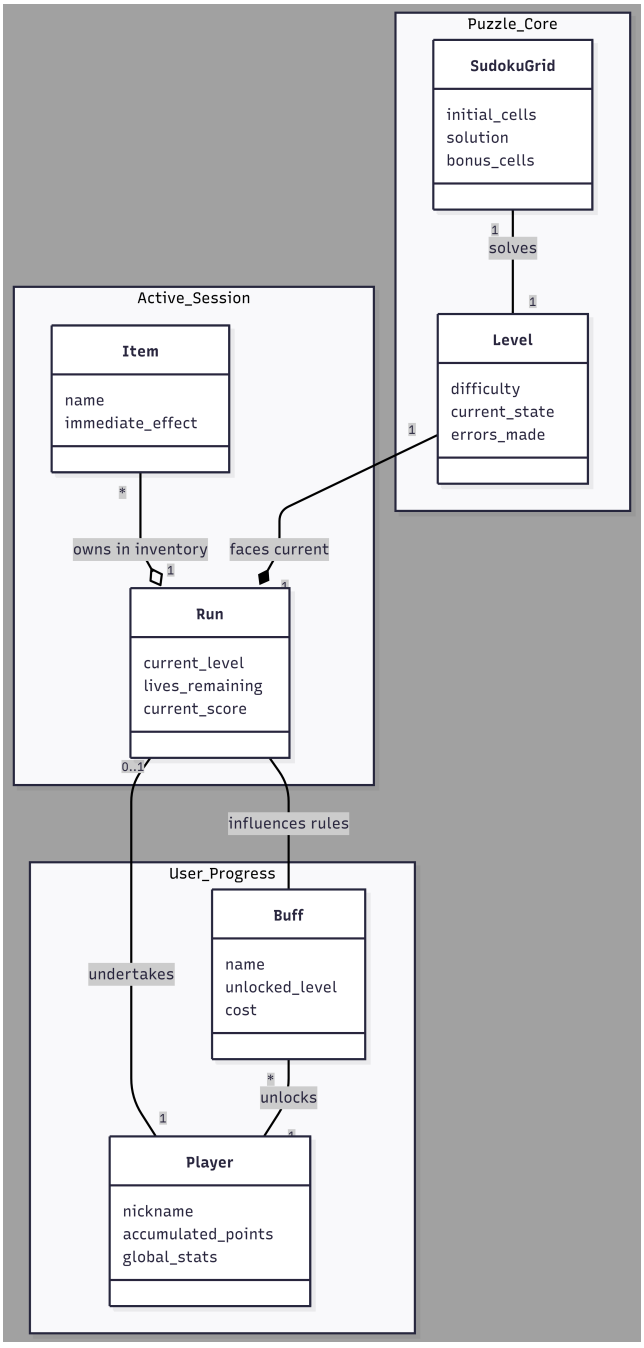

Figura 1.1: Schema UML dell'analisi del dominio, con rappresentate le entità principali ed i rapporti fra loro.

Capitolo 2

Design

2.1 Architettura

L'architettura di PuzzleRogue segue rigorosamente il pattern **Model-View-Controller (MVC)**. Questa scelta è stata dettata dalla necessità di separare nettamente la logica di business (il gioco del Sudoku e la gestione della progressione Roguelike) dalla rappresentazione grafica e dall'interazione utente.

I componenti principali interagiscono come segue:

- **Model**: Rappresenta lo stato del gioco e la logica di business. Include le entità del dominio (`Run`, `User`, `SudokuGrid`), i servizi applicativi (`RunService`, `SudokuGenerator`) e lo strato di persistenza (`DAO`). Il modello è completamente indipendente dalla vista e non ha alcuna dipendenza verso i package JavaFX.
- **View**: Gestisce l'interfaccia grafica e la presentazione dei dati all'utente. È composta da file FXML e classi di supporto per la gestione degli asset grafici. La View osserva lo stato del Model (spesso indirettamente tramite il Controller) per aggiornarsi.
- **Controller**: Funge da intermediario, gestendo gli eventi di input dell'utente (click, inserimento numeri) e orchestrando le chiamate ai servizi del Model. Al termine delle operazioni, aggiorna la View per riflettere il nuovo stato.

L'interazione è unidirezionale per quanto riguarda le dipendenze: il Controller conosce Model e View, ma il Model non conosce nessuno degli altri due. Questo design garantisce che il cuore logico dell'applicazione possa essere testato isolatamente o riutilizzato con un'interfaccia diversa (es. una CLI o una GUI web) senza modifiche.

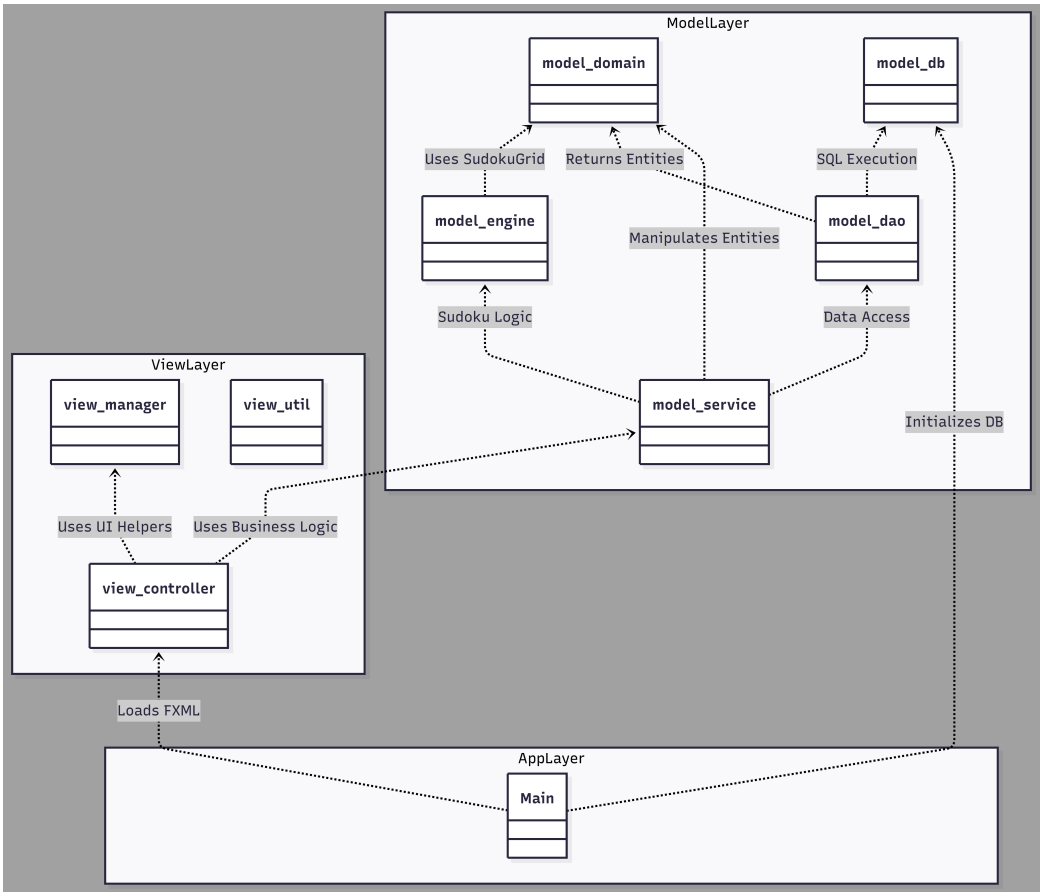

Figura 2.1: Diagramma dei Package che illustra l'architettura MVC e le dipendenze tra i livelli.

2.2 Design Dettagliato

2.2.1 Gestione della Connessione al Database

Problema L'applicazione necessita di un accesso centralizzato e concorrente al database SQLite locale. La creazione di molteplici istanze di connessione o gestori del database potrebbe portare a conflitti di accesso al file, incoerenza dei dati o spreco di risorse. Inoltre, è necessario garantire che lo schema del database sia inizializzato correttamente all'avvio dell'applicazione una sola volta.

Soluzione Si è scelto di utilizzare il **Pattern Singleton** per la classe `DatabaseManager`. Questa classe è responsabile di mantenere l'unica istanza della connessione al database e di fornire metodi per l'accesso ai dati a tutti i DAO (`RunDAO`, `UserDAO`). Il costruttore è privato e l'accesso avviene tramite un metodo statico `getInstance()` che garantisce l'unicità dell'istanza.

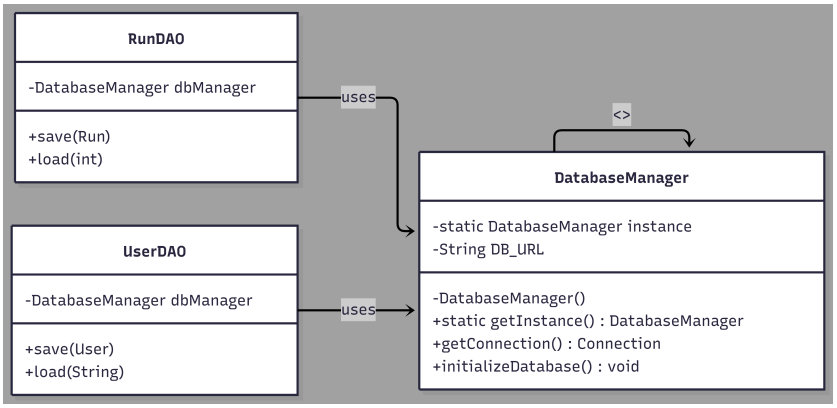

Figura 2.2: Applicazione del pattern Singleton alla classe DatabaseManager.

2.2.2 Creazione ed Estensione dei Potenziamenti

Problema Il gioco prevede diversi tipi di "Buff" (potenziamenti) che il giocatore può sbloccare. Ogni Buff ha proprietà specifiche (es. costo, livelli massimi, descrizione) ma condivide una struttura dati comune. Il sistema deve poter recuperare l'istanza del Buff corretto partendo da un semplice identificativo (es. salvato nel database) per visualizzarlo nello shop o verificarne le proprietà, senza che il client debba conoscere le classi concrete. Inoltre, si vuole facilitare l'aggiunta di nuovi tipi di Buff in futuro senza modificare il codice che li utilizza.

Soluzione È stato implementato il **Pattern Factory** (nella variante Simple Factory / Registry) tramite la classe `BuffFactory`. Questa classe mantiene un registro delle istanze (prototipi) dei Buff disponibili e fornisce un metodo statico `getBuff(BuffType type)` che restituisce l'istanza corretta. Questo centralizza la definizione dei metadati di gioco.

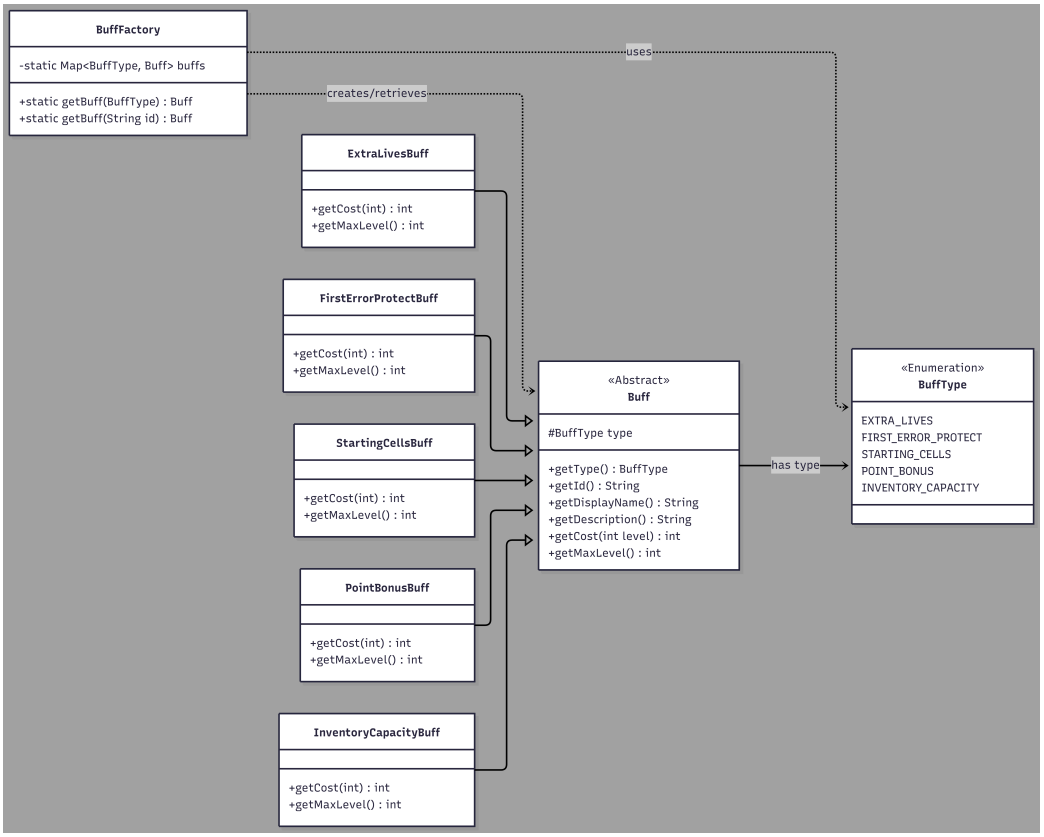

Figura 2.3: Uso del pattern Factory per la gestione centralizzata dei Buff.

2.2.3 Flusso di Navigazione

Problema L'applicazione presenta un flusso di navigazione complesso, con stati che dipendono dal contesto dell'utente (es. "Utente non loggato", "In partita", "Menu principale"). È fondamentale gestire le transizioni tra le schermate in modo coerente, impedendo ad esempio di accedere alla partita se non si è selezionato un personaggio, o di tornare al menu senza salvare.

Soluzione Il flusso è stato modellato come una macchina a stati finiti, dove ogni "Schermata" (View + Controller) rappresenta uno stato. Un `ViewManager` centrale orchestra le transizioni. Il diagramma seguente illustra gli stati permessi e gli eventi che scatenano le transizioni.

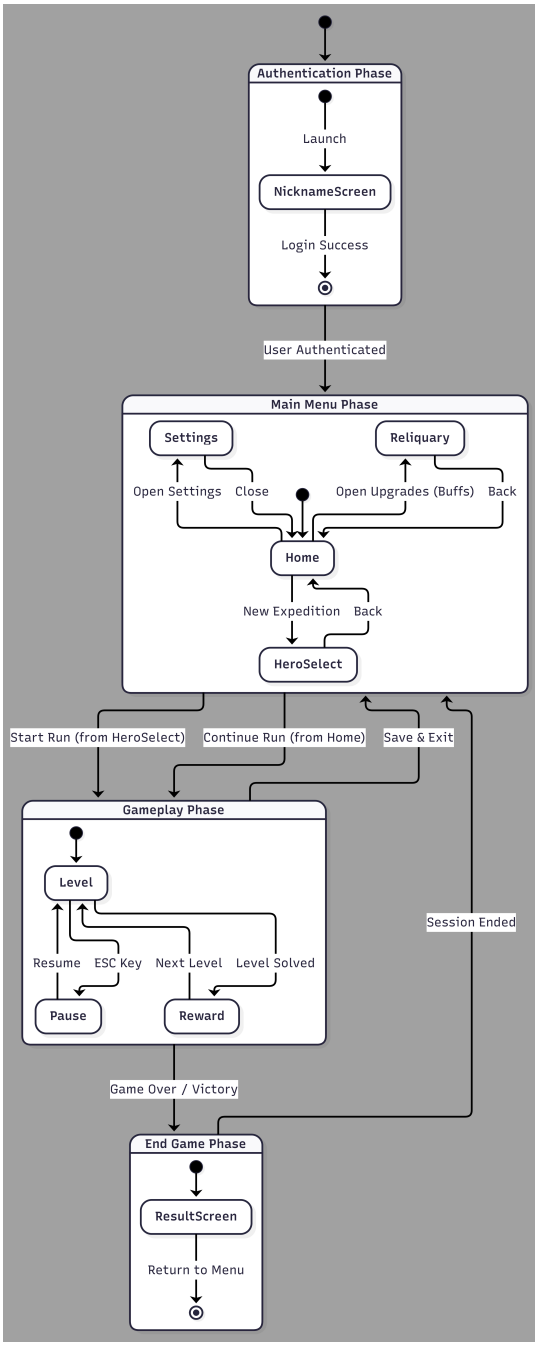

Figura 2.4: Diagramma degli stati che modella la navigazione dell'interfaccia utente.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il progetto adotta una strategia di testing automatizzato focalizzata sulla logica di business e sulla persistenza dei dati, componenti critici per l'integrità dell'esperienza di gioco. Per l'implementazione della suite di test è stato utilizzato il framework **JUnit 5**.

La copertura dei test include:

- **Core Logic**: Verifica della corretta generazione delle griglie Sudoku (`SudokuGenerator`) e delle regole di validazione delle mosse (`SudokuEngine`).
- **Servizi**: Test dei servizi principali come `RunService` e `PointService` per garantire il corretto calcolo dei punteggi e la gestione dei Buff.
- **Persistenza**: Test di integrazione per i DAO (`RunDAO`, `UserDAO`) per verificare il corretto salvataggio e recupero dello stato di gioco e dell'inventario dal database SQLite.

I test sono completamente automatizzati e possono essere eseguiti tramite il task standard di Gradle `test`.

3.2 Note di sviluppo

Cristian Postovan

Responsabilità: Logica Core (Puzzle Engine), Logica Rogue-like, GUI di Gioco.

- **Java Streams API**
  - Utilizzati per la manipolazione concisa e dichiarativa di array e collezioni, in particolare per la copia profonda (deep copy) delle matrici di gioco durante la fase di generazione procedurale.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/model/service/SudokuGenerator.java#L125`
- **Gestione Risorse Audio (Java Sound API)**
  - Implementazione di un gestore audio centralizzato (`SoundManager`) che utilizza `Clip` e `AudioSystem` per il caricamento e la riproduzione asincrona di effetti sonori e musica di sottofondo, con gestione del volume e caching delle risorse.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/view/manager/SoundManager.java#L12-L30`
- **JavaFX Event Filtering & Binding**
  - Utilizzo avanzato della gestione eventi di JavaFX (Event Filters) per intercettare click globali e gestire la deselezione delle celle, e dei Binding per il ridimensionamento responsivo dell'interfaccia.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/view/controller/GameController.java#L113-L123`

Tommaso Conti

Responsabilità: Persistenza, Gestione Account, Sistema Buff, GUI di Sistema.

- **Java Text Blocks (Multilines Strings)**
  - Utilizzati per scrivere query SQL complesse e parametrizzate direttamente nel codice Java, migliorando significativamente la leggibilità e la manutenibilità rispetto alla concatenazione di stringhe tradizionale.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/model/dao/RunDAO.java#L38-L41`
- **Static Initializer Blocks**
  - Impiegati nel `BuffFactory` per la registrazione statica delle strategie di gioco (Buff) all'interno di una `EnumMap`, permettendo l'associazione efficiente e type-safe tra enumerativi e implementazioni concrete senza overhead a runtime.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/model/domain/buff/BuffFactory.java#L13-L19`
- **JDBC Advanced Features (Generated Keys)**
  - Utilizzo della funzionalità `RETURN_GENERATED_KEYS` per il recupero atomico delle chiavi primarie autogenerate durante l'inserimento, garantendo la sincronizzazione immediata tra oggetti in memoria e record database senza query aggiuntive.
  - Permalink: `https://github.com/Yogghevole/puzzleRogue/blob/main/src/main/java/model/dao/RunDAO.java#L44-L62`

In merito al codice riadattato:

- L'algoritmo di generazione del Sudoku (backtracking) implementato in `SudokuGenerator` è una variazione di algoritmi standard di risoluzione dei constraint satisfaction problems, adattato per supportare la rimozione di celle in base alla difficoltà desiderata.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Cristian Postovan

Il lavoro svolto si è concentrato principalmente sul core del gameplay e sull'esperienza utente diretta, curando nel dettaglio non solo la logica ma anche l'aspetto estetico e sonoro.

- **Punti di forza**: L'implementazione dell'engine di Sudoku è risultata robusta e performante, permettendo la generazione di puzzle validi in tempi impercettibili. Grande attenzione è stata dedicata alla selezione e integrazione degli asset grafici e sonori per creare un'atmosfera coerente; molte risorse visive sono state generate tramite strumenti di AI e successivamente rielaborate con Photoshop per adattarle allo stile "Darkest Dungeon" del gioco. L'integrazione con JavaFX tramite binding ha reso l'interfaccia reattiva e fluida.

- **Debolezze**: La gestione degli eventi nella griglia di gioco ha richiesto diverse iterazioni per bilanciare correttamente la logica di selezione e input da tastiera (gestendo sia click che shortcut), portando a una complessità del codice di controllo forse riducibile con un approccio a macchina a stati più formale fin dall'inizio. Anche la ricerca e l'adattamento degli asset grafici ha richiesto un tempo significativo, sottraendolo parzialmente al refactoring del codice.

- **Ruolo nel gruppo**: Mi sono occupato della progettazione della logica di dominio (SudokuEngine), della realizzazione dell'interfaccia di gioco principale e della direzione artistica (scelta immagini, suoni, UI experience), assicurando che le regole del Sudoku e le meccaniche Rogue-like fossero integrate in un'esperienza utente coesa.

Per quanto riguarda gli sviluppi futuri, oltre a un necessario ribilanciamento della difficoltà basato su feedback reali, il passo fondamentale sarebbe la sostituzione completa degli asset "placeholder" (ispirati a Darkest Dungeon) con grafica originale proprietaria. Questo, unito a un'ottimizzazione del database, permetterebbe la pubblicazione del gioco su store digitali come Steam.

Tommaso Conti

Il mio contributo si è focalizzato sull'infrastruttura di supporto al gioco, sulla persistenza dei dati e sul sistema di progressione.

- **Punti di forza**: Il sistema di persistenza basato su SQLite e DAO garantisce un salvataggio affidabile dello stato di gioco. Il design pattern Factory applicato ai Buff ha reso il sistema di potenziamenti modulare ed estensibile: aggiungere nuovi buff o malus richiede modifiche minime e isolate al codice esistente.

- **Debolezze**: L'uso di JDBC puro ha comportato una certa verbosità nel codice di accesso ai dati; in un contesto professionale l'adozione di un ORM avrebbe snellito il layer di persistenza. Alcune transizioni nell'interfaccia dei menu potrebbero beneficiare di animazioni più curate per allinearsi alla fluidità del gameplay principale.

- **Ruolo nel gruppo**: Ho gestito l'intero stack di persistenza (DatabaseManager, DAO), la logica di gestione utente e sessione, e il sistema di meta-progressione (Buffs, Shop), oltre a curare la GUI di sistema (Home, Menu, Legacy).

Come sviluppi futuri, sarebbe interessante implementare un sistema di cloud save per la sincronizzazione cross-device e introdurre un sistema di "Achievement" strutturato basato su eventi complessi. Inoltre, il motore di gioco potrebbe essere esteso per supportare altre varietà di puzzle logici oltre al Sudoku, come il Kakuro (Cross Sums), aumentando notevolmente la varietà del gameplay.

Appendice A

Guida utente

A.1 Avvio dell'applicazione

Per avviare il gioco è sufficiente eseguire il file JAR distribuito. Assicurarsi di avere installato una versione del JRE compatibile (Java 17 o superiore).

Appena avviata l'applicazione, verrà richiesto all'utente di inserire un **Nickname**:

- Se il nickname inserito è già presente nel database, il sistema caricherà automaticamente l'account esistente recuperando tutti i progressi (buff sbloccati, eventuale run in corso).
- Se il nickname non è presente, verrà creato istantaneamente un nuovo profilo utente associato a quel nome.

Una volta effettuato l'accesso, verrà mostrata la schermata *Home*.

A.2 Menu Principale

Dalla schermata iniziale (Home) sono disponibili le seguenti opzioni principali, presentate nell'ordine visualizzato:

- **Venture Forth** (Continua Spedizione): Riprende l'ultima partita salvata esattamente dal punto in cui era stata interrotta, mantenendo lo stato della griglia, le vite e l'inventario.
- **New Expedition** (Nuova Spedizione): Avvia una nuova partita (Run). Verrà iniziata una nuova sessione di gioco.

- **The Ancestor's Legacy** (Reliquiario/Buffs): Apre il menu di gestione dei potenziamenti permanenti. Qui è possibile visualizzare e gestire i buff attivi o sbloccati che forniscono vantaggi strategici.

In alto a destra è presente un'icona a forma di **ingranaggio** che apre il pannello delle impostazioni globali. Da qui è possibile:

- Regolare il volume della **Musica** e degli **Effetti Sonori** (SFX).
- Effettuare il **Log Out** per cambiare utente.
- **Chiudere il gioco** (Exit Game).

A.3 Controlli di Gioco

L'interfaccia di gioco è progettata per essere utilizzata intuitivamente tramite mouse e tastiera.

A.3.1 Interazione con la Griglia

- **Selezione Cella**: Cliccare con il tasto sinistro del mouse su una cella della griglia Sudoku per selezionarla. La cella selezionata verrà evidenziata.
- **Inserimento Numero**: Con una cella selezionata, premere i tasti numerici da `1` a `9` sulla tastiera o selezionare il numero dalla barra inferiore.
- **Cancellazione**: Per cancellare il contenuto di una cella, utilizzare l'icona della **Gomma** situata nel menu in basso, vicino alla lista dei numeri. È possibile rimuovere solo i numeri errati o le note.

A.3.2 Modalità Appunti (Note Mode)

È possibile inserire "note" (piccoli numeri candidati) in una cella per aiutarsi nella risoluzione logica senza commettere errori.

- **Attivazione/Disattivazione**: Cliccare sull'icona della **Matita**, situata in basso vicino alla gomma, per attivare o disattivare la modalità appunti.
- Quando la modalità appunti è attiva, i numeri inseriti appariranno in piccolo all'interno della cella come annotazioni. Queste non vengono conteggiate come mosse definitive, quindi non causano la perdita di vite in caso di "errore" (non vengono validate contro la soluzione).

A.4 Meccaniche di Gioco

- **Obiettivo**: Risolvere correttamente la griglia Sudoku per completare il livello attuale (piano del dungeon) e avanzare al successivo.
- **Vite ed Errori**: Il giocatore dispone di un numero limitato di vite, visualizzate nell'interfaccia. Ogni volta che si inserisce un numero definitivo errato (non corrispondente alla soluzione), si perde una vita. Se le vite scendono a zero, la partita termina (Game Over).
- **Inventario e Oggetti**: Durante la partita, in **alto a destra**, sono visibili gli slot inventario. Cliccando sugli oggetti disponibili è possibile attivarne l'effetto (es. recupero vite o aiuti sulla griglia).
- **Pausa e Menu In-Game**: È possibile mettere in pausa il gioco e accedere al menu delle opzioni tramite l'icona dell'ingranaggio in alto a destra.

A.4.1 Progressione e The Ancestor's Legacy

Al termine di ogni partita (sia in caso di vittoria che di sconfitta), il giocatore guadagna dei **Punti** calcolati in base alle prestazioni ottenute durante la run. Questi punti vengono accumulati nel saldo dell'account e possono essere spesi nel menu **The Ancestor's Legacy** (dalla Home) per sbloccare e potenziare **Buff Permanenti**.

I buff acquistati influenzeranno positivamente tutte le partite future. Attualmente sono disponibili i seguenti potenziamenti:

- **Extra Lives**: Aumenta il numero di vite iniziali disponibili all'inizio di ogni partita.
- **First Error Protect**: Protegge dal primo errore commesso in ogni livello, evitando la perdita di una vita.
- **Starting Cells**: Aumenta il numero di celle già rivelate all'inizio di ogni livello, facilitandone la risoluzione.
- **Point Bonus**: Incrementa il moltiplicatore dei punti ottenuti a fine run.
- **Inventory Capacity**: Espande la capacità dell'inventario, permettendo di trasportare un maggior numero di oggetti consumabili.

A.5 Salvataggio

Per salvare la partita è necessario aprire il menu delle impostazioni tramite l'icona dell'ingranaggio in alto a destra (identica a quella del menu Home). All'interno di questo menu, è presente il pulsante **"Save & Exit"** che permette di salvare i progressi correnti e uscire dalla run, tornando al menu principale.