Maintenance: Sala Giochi Virtuale

Progetto: Sala Giochi Virtuale
Data: 13 Gennaio 2026
Team di Progetto: Team UNI
Corso: Ingegneria del Software – Università degli Studi di Bergamo

Contenuto:

- Software Maintenance

1. Introduzione

1.1 Reverse Engineering

Il progetto "Sala Giochi" si può classificare come sviluppo *Greenfield* (sviluppato da zero), motivo per cui non sono state necessarie attività di Reverse Engineering. Il team non ha dovuto analizzare o ricostruire la documentazione di sistemi *legacy* preesistenti, in quanto l'intero codice sorgente è stato scritto ex-novo partendo dai requisiti definiti nelle fasi iniziali.

2. Attività di Refactoring

Le attività di refactoring sono state condotte in due modalità distinte: interventi architetturali mirati (per risolvere difetti di progettazione) e interventi continui (durante la scrittura del codice).

2.1 Refactoring Architetturale: Decomposizione della *God Class* BattleShip

- **Problema rilevato (*Code Smell*):** *God Class / Large Class*. Nella prima iterazione di sviluppo, la classe `BattleShip.java` accentrava troppe responsabilità: gestiva la logica della partita, il disegno della griglia grafica (JavaFX), la logica di attacco, il posizionamento delle navi. Tale approccio monolitico comprometteva la coesione del codice e ostacolava fortemente le attività di manutenzione e testing.

- **Soluzione Applicata (Delegation Pattern):** Il team ha applicato il Delegation Pattern, scomponendo la classe monolitica in componenti più piccoli e coesi:
    - La gestione della griglia grafica è stata isolata.
    - La logica di posizionamento delle navi e l'IA del computer sono state delegate a metodi o classi di supporto specifiche.
    - La classe `BattleShip` principale ora funge da coordinatore, delegando i compiti specifici ai sotto-componenti.
- **Beneficio:** aumento della coesione e riduzione dell'accoppiamento. Ora è possibile modificare l'algoritmo dell'IA senza rischiare di rompere la visualizzazione grafica.

Nota: un simile approccio di *refactoring* è stato applicato alla classe *Roulette.java*, tramite le tecniche accennate nel documenti dei Design.

2.2 Refactoring Continuo (Pair Programming)

Molte attività di "pulizia del codice" (*Clean Code*) non sono state pianificate come task separati, ma svolte in tempo reale durante le sessioni di **Pair Programming**. La presenza costante di un *Navigator* a fianco del *Driver* ha permesso di applicare micro-refactoring istantanei (*Refactor as you go*):

- **Rename Method/Available:** rinominare variabili e metodi con nomi più esplicativi nel momento stesso in cui venivano scritti o riutilizzati, migliorando la leggibilità immediata.
- **Dead Code Elimination:** rimozione immediata di porzioni di codice commentato o inutilizzato prima del commit.

Questa modalità ha ridotto drasticamente il debito tecnico accumulato, rendendo il codice "pulito" fin dalla prima stesura condivisa.

3. Debito Tecnico Residuo

Nonostante l'attenzione alla qualità, alcune scelte dettate dai vincoli temporali costituiscono un debito tecnico noto da affrontare in future operazioni di manutenzione:

1. **Sicurezza Password:** Le password sono attualmente salvate in chiaro nel database. Un intervento prioritario sarebbe l'introduzione di un algoritmo di Hashing per garantire la protezione dei dati.

2. **Internazionalizzazione:** Le stringhe di testo sono legate direttamente al codice. Sarebbe opportuno estrarle in file di risorse (`.properties`) per supportare il multilingua.

La manutenzione generale del software è pensata per poter essere svolta da qualunque programmatore esterno allo sviluppo del codice, e non solo dai propri sviluppatori. A garanzia di quanto affermato vi è una completa documentazione, seguita dall'aggiunta di commenti per i metodi più complicati (o per determinate scelte di programmazione adottate), oltre a porzioni di codice auto-esplicative.