UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

CORSO DI LAUREA IN INFORMATICA

Insegnamento Laboratorio di programmazione ad oggetti

Risiko UnivAQ - by MASTERMINDS

Membri del Team		
Cognome	**Nome**	**Matricola**
MARINUCCI ALESSANDRO	261682	alessandro.marinucci2@student.univaq.it
ODOARDI DAVIDE	292216	davide.odoardi@student.univaq.it
FASCIANO DAVIDE	245433	davide.fasciano@student.univaq.it

A.A. 2003/2024

Sommario

Capitolo 1 – Caso di studio

1.1 – Descrizione dell'applicazione

L'obiettivo di questo progetto è progettare e implementare una versione digitale del famoso gioco da tavolo **Risiko**, utilizzando il linguaggio di programmazione **Java**. L'applicazione sfrutta i concetti fondamentali della **programmazione orientata agli oggetti**, come classi, ereditarietà, polimorfismo e gestione degli eventi, per creare un'esperienza di gioco fedele all'originale.

Caratteristiche principali:

- **Interfaccia Utente:** Basata su **console**, con utilizzo della libreria **Jansi** per migliorare la formattazione e i colori del testo.
- **Numero di Giocatori:** Supporta da **2 a 6 giocatori**, permettendo partite multigiocatore locale.
- **Mappa di Gioco:** Rappresentazione dei **territori** e dei **continenti**, inclusi confini e connessioni, per simulare fedelmente il tabellone di Risiko.
- **Gestione delle Armate:** Implementazione delle meccaniche di distribuzione, attacco, difesa e spostamento delle armate tra territori.
- **Carte e Obiettivi:** Introduzione di **carte territorio**, **carte obiettivo** e **jolly**, che influenzano la strategia di gioco e determinano le condizioni di vittoria.
- **Dadi Virtuali:** Simulazione del lancio dei **dadi** per risolvere le fasi di attacco e difesa tra i giocatori.
- **Stato del Gioco:** Controllo delle diverse fasi del gioco, come la distribuzione iniziale delle armate, le fasi di attacco, rinforzo e spostamento.
- **Persistenza:** Possibilità di **salvare** e **riprendere** una partita in corso, garantendo flessibilità e continuità nell'esperienza di gioco.
- **Log delle Azioni:** Registrazione dettagliata su file di tutte le azioni svolte durante la partita, come attacchi, movimenti e cambi di turno, per permettere un'analisi post-partita.

Tecnologie e Approcci Utilizzati:

- **Programmazione Orientata agli Oggetti:** Uso estensivo di classi e oggetti per modellare le entità del gioco (giocatori, territori, armate, ecc.).
- **Collezioni Java:** Utilizzo di strutture dati come liste, mappe e set per gestire gli elementi dinamici del gioco.
- **Stream ed Espressioni Lambda:** Applicazione di Stream API ed espressioni lambda per operazioni efficienti sulle collezioni.
- **Gestione degli Eventi:** Implementazione di meccanismi per gestire eventi di gioco, come attacchi e cambi di turno.
- **Modularità del Codice:** Suddivisione del codice in packages distinti (core, datamodel, service, impl, utils) per una migliore organizzazione e manutenzione.
- **Persistenza dei Dati:** Salvataggio e caricamento dello stato del gioco utilizzando la serializzazione degli oggetti.
- **Gestione dei Colori e Formattazione del Testo:** Utilizzo della libreria **Jansi** per migliorare l'esperienza utente nell'interfaccia console.

1.2 – Funzionalità

L'applicazione è progettata per offrire un'esperienza di gioco completa di Risiko, implementando tutte le funzionalità chiave del gioco originale. Di seguito è riportata una lista completa dei requisiti funzionali, organizzati per tipologie di utenti:

Tipologie di Utenti:

- **Giocatori Umani:** Partecipano al gioco inserendo comandi tramite l'interfaccia console.
- **Amministratore di Gioco (Sistema):** Gestisce le logiche interne del gioco, come la distribuzione delle armate e il controllo delle fasi.

Funzionalità per i Giocatori Umani:

1. **Creazione e Configurazione della Partita:**
    - Inserimento del nome del giocatore.
    - Scelta del colore rappresentativo.
    - Selezione del numero di giocatori (da 2 a 6).
2. **Gestione delle Armate:**
    - Distribuzione iniziale delle armate sui territori assegnati.
    - Posizionamento strategico delle armate durante le fasi di rinforzo.
    - Scambio di carte per ottenere armate aggiuntive.
3. **Movimenti e Attacchi:**
    - Attacco a territori avversari adiacenti.
    - Utilizzo di dadi virtuali per determinare l'esito degli attacchi e delle difese.
    - Spostamento di armate tra territori controllati.
4. **Gestione delle Carte:**
    - Ricezione di carte territorio dopo la conquista di almeno un territorio nel turno.
    - Scambio di combinazioni di carte per ottenere rinforzi.
    - Visualizzazione delle carte possedute.
5. **Obiettivi di Gioco:**
    - Assegnazione di un obiettivo segreto a ogni giocatore.
    - Possibilità di visualizzare l'obiettivo durante il gioco.
    - Controllo del raggiungimento dell'obiettivo per determinare la vittoria.
6. **Interazione con l'Interfaccia Utente:**
    - Inserimento di comandi tramite console per eseguire azioni.
    - Visualizzazione dello stato attuale della mappa, delle armate e dei territori controllati.
    - Accesso a un menu per salvare la partita, visualizzare informazioni o terminare il gioco.

Funzionalità per l'Amministratore di Gioco (Sistema):

1. **Inizializzazione della Partita:**
    - Creazione della mappa con territori e continenti.
    - Distribuzione casuale dei territori tra i giocatori.
    - Calcolo del numero di armate iniziali in base al numero di giocatori.

2. **Gestione delle Fasi di Gioco:**
    - Controllo delle fasi: distribuzione armate, attacco, spostamento.
    - Gestione del turno dei giocatori e dell'ordine di gioco.
    - Monitoraggio delle condizioni di vittoria.
3. **Gestione delle Carte e del Mazzo:**
    - Creazione del mazzo di carte territorio e obiettivo.
    - Distribuzione delle carte obiettivo ai giocatori.
    - Gestione del mazzo di pesca e delle carte scartate.
4. **Persistenza e Salvataggio:**
    - Salvataggio dello stato della partita su file per permettere di riprendere il gioco in un secondo momento.
    - Caricamento di una partita precedentemente salvata.
5. **Log delle Azioni:**
    - Registrazione su file di tutte le azioni svolte durante la partita (es. attacchi, movimenti, distribuzioni).
    - Creazione di un log dettagliato per analisi o debugging.
6. **Gestione degli Eventi di Gioco:**
    - Calcolo dei rinforzi basato sul controllo di territori e continenti.
    - Verifica delle regole durante gli attacchi e le difese.
    - Gestione delle eliminazioni dei giocatori e della fine della partita.

1.3 – Domain Model

Il domain model dell'applicazione è rappresentato da un insieme di classi che modellano le entità fondamentali del gioco Risiko e le relazioni tra di esse. Di seguito è fornita una descrizione delle principali classi e delle loro interazioni.

Classi Principali:

1. **Gioco:**
    - Rappresenta lo stato globale della partita.
    - Contiene informazioni sui giocatori, sulla mappa, sul mazzo di carte e sullo stato del turno.
    - Gestisce le fasi del gioco e la progressione dei turni.
2. **Giocatore:**
    - Modella un partecipante al gioco.
    - Contiene informazioni come nome, colore, armate totali, territori controllati, carte possedute e obiettivo assegnato.
    - Gestisce le azioni che il giocatore può compiere durante il turno.
3. **Mappa:**
    - Rappresenta il tabellone di gioco.
    - Composta da una lista di continenti.
4. **Continente:**
    - Rappresenta un continente sulla mappa.
    - Contiene una lista di territori e informazioni sui confini.
5. **Territorio:**

- Modella un territorio specifico.
- Contiene informazioni sul nome, il giocatore che lo controlla, il numero di armate presenti e i territori adiacenti.
- Permette di gestire le armate e le interazioni con altri territori.

6. **Carta:**
    - Classe base per le carte del gioco.
    - Ha sottoclassi specifiche come **CartaTerritorio** e **CartaObiettivo**.
7. **CartaTerritorio:**
    - Associa un territorio a una figura (es. cannone, fante, cavaliere).
    - Utilizzata per scambiare carte e ottenere rinforzi.
8. **CartaObiettivo:**
    - Contiene la descrizione di un obiettivo segreto assegnato al giocatore.
    - Definisce le condizioni di vittoria per il giocatore.
9. **MazzoDiCarte:**
    - Gestisce il mazzo delle carte territorio.
    - Permette di pescare carte e gestisce le carte scartate.
10. **TurnoGioco:**
    - Contiene lo stato corrente del turno di un giocatore.
    - Gestisce la fase attuale del turno (inizio, distribuzione armate, menu, fine turno) e altre informazioni come le armate da distribuire.

Relazioni tra le Classi:

- **Gioco** aggrega **Giocatore**, **Mappa**, **MazzoDiCarte** e **TurnoGioco.**.
- **Giocatore** possiede una lista di **Territorio** (territori controllati) e una lista di **Carta** (carte possedute).
- **Mappa** contiene una lista di **Continente**, che a loro volta contengono una lista di **Territorio**.
- **Territorio** ha riferimenti ai **Territorio** adiacenti e al **Giocatore** che lo controlla.
- **CartaTerritorio** e **CartaObiettivo** sono specializzazioni della classe base **Carta**.
- **MazzoDiCarte** gestisce una lista di **Carta**.

Capitolo 2 – Implementazione

2.1 – Struttura dei packages

L'applicazione è organizzata in una serie di packages per migliorare la modularità e la manutenibilità del codice. La struttura dei packages è la seguente:

- **it.univaq.disim.lpo.risiko.core:** Contiene le classi che tutti gli elementi fondamentali del gioco di Risiko, ossia le classi e i componenti necessari per avviare, gestire e coordinare la logica di gioco.
- **it.univaq.disim.lpo.risiko.core.model**: Contiene le classi che rappresentano i dati e le entità del dominio (es. Giocatore, Mappa, Territorio, Carta, ecc.).
- **it.univaq.disim.lpo.risiko.core.service**: Definisce le interfacce dei servizi che gestiscono la logica di business.
- **it.univaq.disim.lpo.risiko.core.service.impl**: Contiene le implementazioni delle interfacce dei servizi.
- **it.univaq.disim.lpo.risiko.core.utils**: Include classi di utilità utilizzate in tutta l'applicazione.

2.2 – Descrizione delle Classi e dei Metodi

Di seguito è fornita una descrizione dettagliata delle principali classi e dei metodi implementati, suddivisi per package.

2.2.1 – Package **Core**

Classe StartGame:

- **Descrizione:** Classe di avvio dell'applicazione,questa classe fornisce il punto d'ingresso del programma, limitandosi a richiamare il metodo statico di avvio del gioco nella classe Runner.In questo modo, la logica di esecuzione principale del gioco rimane nella classe Runner, mentre StartGame funge solo da "bootstrap" dell'applicazione.

Classe Runner:

- **Descrizione:** Funge da punto d'ingresso principale per l'inizializzazione e la gestione del flusso del gioco. Imposta le configurazioni necessarie della console,coordina le istanze del servizio ed esegue il ciclo di gioco principale fino alla conclusione.
- **Le responsabilità di questa classe includono:**
- Installare e disinstallare il supporto della console ANSI per l'output colorato.
- Inizializzare o caricare una sessione di gioco tramite GiocoService.
- Distribuire gli eserciti iniziali ai giocatori tramite GiocatoreService.
- Eseguire i turni in un ciclo fino alla fine del gioco o al ritorno dell'utente al menu.
- Registrare risultati e decisioni importanti per l'audit e il debug.

Classe RisikoException:

- **Descrizione:** Eccezioni personalizzate per l'applicazione.Viene sollevata quando si verificano condizioni di errore specifiche all'interno del flusso di esecuzione del gioco, consentendo di gestire in modo concentrato problemi imprevisti o situazioni anomale.

Classe InputManagerSingleton:

- **Descrizione:** Classe singleton che gestisce l'input dell'utente tramite **Scanner**.
- **Attributi Principali:**
  - **scanner**: Oggetto **Scanner** per la lettura dell'input.
  - **instance**: Istanza singleton della classe.
- **Metodi Principali:**
  - **readInteger()**: Legge un intero dall'input.
  - **readIntegerUntilPossibleValue(Integer[] possibleValues)**: Legge un intero e verifica che sia tra i valori possibili.
  - **readString()**: Legge una stringa dall'input.
  - **disposeScanner()**: Chiude lo scanner.
  - **close()**: Implementazione del metodo **close** per interfaccia **Closeable**.

2.2.2 – Package **Model**

Classe Gioco:

- **Descrizione:** Rappresenta lo stato globale della partita, inclusi i giocatori, la mappa, il mazzo di carte e lo stato del turno.
- **Attributi Principali:**
  - **fase**: La fase corrente del gioco.
  - **giocatori**: Lista dei giocatori partecipanti.
  - **mappa**: La mappa di gioco.
  - **mazzoDiCarte**: Il mazzo di carte territorio.
  - **currentTurnState**: Lo stato corrente del turno.
  - **roundCount**: Il numero di turni trascorsi.
  - **logFileName**: Il nome del file di log corrente.
  - **partitaInCorso**: Indica se la partita è in corso.
  - **loadedGame**: Indica se la partita è stata caricata da un salvataggio.
  - **ritornaAlMenu**: Indica se il gioco deve tornare al menu principale.
- **Metodi Principali:**
  - Costruttore per inizializzare un nuovo gioco.
  - Getter e setter per gli attributi.

Classe Giocatore:

- **Descrizione:** Modella un partecipante al gioco, con informazioni sulle armate, i territori controllati, le carte possedute e l'obiettivo assegnato.
- **Attributi Principali:**
    - **nome**: Il nome del giocatore.
    - **colore**: Il colore delle armate del giocatore.
    - **armate**: Il numero totale di armate possedute.
    - **territoriControllati**: Lista dei territori controllati.
    - **carte**: Lista delle carte possedute.
    - **obiettivo**: L'obiettivo segreto assegnato.
    - **haRicevutoCartaBonus**: Indica se il giocatore ha ricevuto una carta bonus nel turno corrente.
    - **territoriConquistatiNelTurno**: Numero di territori conquistati nel turno.
- **Metodi Principali:**
    - Metodi per aggiungere e rimuovere territori.
    - Metodi per aggiungere e rimuovere carte.
    - Metodi per gestire le armate.
    - Getter e setter per gli attributi.

Classe Mappa:

- **Descrizione:** Rappresenta il tabellone di gioco, contenente i continenti e i territori.
- **Attributi Principali:**
    - **continenti**: Lista dei continenti sulla mappa.
- **Metodi Principali:**
    - Metodi per ottenere i continenti e i territori.
    - Getter e setter per gli attributi.

Classe Continente:

- **Descrizione:** Modella un continente sulla mappa, contenente una lista di territori.
- **Attributi Principali:**
    - **nome**: Il nome del continente.
    - **territori**: Lista dei territori nel continente.
- **Metodi Principali:**
    - Getter e setter per gli attributi.

Classe Territorio:

- **Descrizione:** Modella un territorio specifico, con informazioni sul giocatore che lo controlla, il numero di armate e i territori adiacenti.
- **Attributi Principali:**
    - **nome**: Il nome del territorio.
    - **giocatore**: Il giocatore che controlla il territorio.
    - **armate**: Il numero di armate presenti nel territorio.
    - **territoriAdiacenti**: Lista dei territori adiacenti.
    - **continente**: Il continente a cui appartiene il territorio.
- **Metodi Principali:**
    - Metodi per aggiungere e rimuovere armate.

- Metodi per gestire i territori adiacenti.
- Getter e setter per gli attributi.

Classe Carta (astratta):

- **Descrizione:** Classe base per le carte del gioco.
- **Attributi Principali:**
  - **tipo**: Il tipo di carta (Cannone, Fante, Cavaliere, Jolly).
  - **territorio**: Il territorio associato alla carta (può essere null).
- **Metodi Principali:**
  - Getter e setter per gli attributi.

Classe CartaTerritorio:

- **Descrizione:** Estende la classe **Carta**, rappresenta una carta territorio specifica.
- **Attributi Principali:**
  - Eredita gli attributi dalla classe **Carta**.
- **Metodi Principali:**
  - Costruttore per inizializzare la carta con un territorio e un tipo.

Classe CartaObiettivo:

- **Descrizione:** Rappresenta un obiettivo segreto assegnato al giocatore.
- **Attributi Principali:**
  - **descrizione**: La descrizione dell'obiettivo.
- **Metodi Principali:**
  - Costruttore per inizializzare l'obiettivo con una descrizione.
  - Getter per la descrizione.

Classe MazzoDiCarte:

- **Descrizione:** Gestisce il mazzo delle carte territorio.
- **Attributi Principali:**
  - **carte**: Una **LinkedList** di carte disponibili nel mazzo.
- **Metodi Principali:**
  - **pescaCarta()**: Permette di pescare una carta dal mazzo.
  - **restituisciCarte(List<Carta> carte)**: Restituisce le carte scambiate al mazzo.
  - **inizializzaMazzo(List<Territorio> territori)**: Inizializza il mazzo con le carte territorio e jolly.

Classe TurnoGioco:

- **Descrizione:** Contiene lo stato corrente del turno di un giocatore.
- **Attributi Principali:**
  - **currentPhase**: La fase attuale del turno (START_TURN, DISTRIBUTE_ARMIES, MENU, END_TURN).
  - **armateDaDistribuire**: Il numero di armate da distribuire nel turno.
  - **turnoTerminato**: Indica se il turno è terminato.
  - **armateTotali**: Il totale delle armate ricevute all'inizio del turno.

    - **numeroTerritori**: Il numero di territori controllati all'inizio del turno.
    - **numeroContinenti**: Il numero di continenti controllati all'inizio del turno.
- **Metodi Principali:**
    - Getter e setter per gli attributi.

2.2.3 – Package **Service**

Questo package contiene le interfacce che definiscono i servizi principali dell'applicazione.

Interfaccia GiocoService:

- **Descrizione:** Definisce i metodi per la gestione della partita.
- **Metodi Principali:**
    - **inizializzaPartita()**: Inizializza una nuova partita.
    - **turnoGiocatore(Giocatore giocatore, Gioco gioco)**: Gestisce il turno di un giocatore.
    - **verificaVittoria(Giocatore giocatore, Gioco gioco)**: Verifica se un giocatore ha vinto.
    - **caricaGioco(String fileName)**: Carica una partita salvata.

Interfaccia GiocatoreService:

- **Descrizione:** Definisce i metodi per la gestione dei giocatori.
- **Metodi Principali:**
    - **creaGiocatori(int numeroGiocatori)**: Crea i giocatori della partita.
    - **distribuzioneTerritori(List<Giocatore> giocatori, Mappa mappa)**: Distribuisce i territori tra i giocatori.
    - **lancioDadiPerPrimoGiocatore(List<Giocatore> giocatori)**: Determina l'ordine dei giocatori.
    - **scegliColoriGiocatori(List<Giocatore> giocatori)**: Permette ai giocatori di scegliere il colore.
    - **distribuzioneInizialeArmate(List<Giocatore> giocatori, int armatePerGiocatore)**: Gestisce la distribuzione iniziale delle armate.
    - **possiedeAlmenoUnaCombinazioneValida(Giocatore giocatore)**: Verifica se il giocatore ha almeno una combinazione valida di carte.
    - **scambiaCartePerArmate(Giocatore giocatore, Gioco gioco)**: Permette al giocatore di scambiare carte per ottenere armate aggiuntive.

Interfaccia MappaService:

- **Descrizione:** Definisce i metodi per la gestione della mappa.
- **Metodi Principali:**
    - **getMappa()**: Restituisce la mappa corrente.

Interfaccia CartaObiettivoService:

- **Descrizione:** Definisce i metodi per la gestione delle carte obiettivo.
- **Metodi Principali:**
    - **generaObiettiviCasuali(int numeroObiettivi)**: Genera una lista di obiettivi casuali.

Interfaccia FileService:

- **Descrizione:** Definisce i metodi per la gestione dei file (salvataggio e caricamento del gioco, logging).
- **Metodi Principali:**
    - **salvaGioco(Gioco gioco, String fileName)**: Salva lo stato del gioco su file.
    - **caricaGioco(String fileName)**: Carica lo stato del gioco da file.
    - **writeLog(String data)**: Scrive un messaggio nel file di log corrente.

2.2.4 – Package **Service.impl**

Questo package contiene le implementazioni delle interfacce dei servizi.

Classe GiocoServiceImpl:

- **Descrizione:** Implementa i metodi definiti in **GiocoService** per gestire la partita.
- **Attributi Principali:**
    - **fileService**: Istanza di **FileServiceImpl** per gestire i file.
    - **giocatoreService**: Istanza di **GiocatoreServiceImpl** per gestire i giocatori.
    - **obiettivoService**: Istanza di **CartaObiettivoServiceImpl** per gestire gli obiettivi.
    - **mappaService**: Istanza di **MappaServiceImpl** per gestire la mappa.
- **Metodi Principali:**
    - **inizializzaPartita()**: Gestisce l'inizializzazione della partita, inclusa la scelta tra nuova partita o caricamento di una esistente.
    - **turnoGiocatore(Giocatore giocatore, Gioco gioco)**: Gestisce tutte le fasi del turno di un giocatore (distribuzione armate, attacco, spostamento, menu).
    - **verificaVittoria(Giocatore giocatore, Gioco gioco)**: Verifica se un giocatore ha raggiunto il suo obiettivo.
    - **salvaEEsci(Gioco gioco)**: Gestisce il salvataggio del gioco e l'uscita.
    - Metodi ausiliari per gestire le fasi di attacco, spostamento, visualizzazione delle informazioni, ecc.

Classe GiocatoreServiceImpl:

- **Descrizione:** Implementa i metodi definiti in **GiocatoreService** per gestire i giocatori.
- **Attributi Principali:**
    - **random**: Oggetto **Random** per generare numeri casuali.
    - **coloriDisponibili**: Lista dei colori disponibili per i giocatori.
- **Metodi Principali:**
    - **creaGiocatori(int numeroGiocatori)**: Permette ai giocatori di inserire il proprio nome.

- **lancioDadiPerPrimoGiocatore(List<Giocatore> giocatori)**: Determina l'ordine di gioco tramite il lancio dei dadi.
- **distribuzioneTerritori(List<Giocatore> giocatori, Mappa mappa)**: Distribuisce casualmente i territori tra i giocatori.
- **scegliColoriGiocatori(List<Giocatore> giocatori)**: Permette ai giocatori di scegliere il colore delle proprie armate.
- **distribuzioneInizialeArmate(List<Giocatore> giocatori, int armatePerGiocatore)**: Gestisce la distribuzione iniziale delle armate sui territori.
- **possiedeAlmenoUnaCombinazioneValida(Giocatore giocatore)**: Verifica se il giocatore ha combinazioni valide di carte.
- **scambiaCartePerArmate(Giocatore giocatore, Gioco gioco)**: Gestisce lo scambio di carte per ottenere armate aggiuntive.
- Metodi ausiliari per calcolare armate, aggiungere/rimuovere territori e armate, ecc.

Classe MappaServiceImpl:

- **Descrizione:** Implementa i metodi definiti in **MappaService** per gestire la mappa.
- **Attributi Principali:**
  - **mappa**: Istanza della mappa di gioco.
  - **continenti**: Lista dei continenti della mappa.
- **Metodi Principali:**
  - **getMappa()**: Restituisce la mappa inizializzata.
  - **inizializzaContinentiETerritori()**: Crea i continenti e i territori con le relative adiacenze.
  - Metodi ausiliari per impostare le adiacenze tra i territori.

Classe CartaObiettivoServiceImpl:

- **Descrizione:** Implementa i metodi definiti in **CartaObiettivoService** per gestire le carte obiettivo.
- **Metodi Principali:**
  - **generaObiettiviCasuali(int numeroObiettivi)**: Genera una lista casuale di obiettivi in base al numero di giocatori.
  - **assegnaObiettiviCasuali(List<Giocatore> giocatori, List<CartaObiettivo> obiettivi)**: Assegna casualmente gli obiettivi ai giocatori.

Classe FileServiceImpl:

- **Descrizione:** Implementa i metodi definiti in **FileService** per gestire il salvataggio, il caricamento e il logging.
- **Attributi Principali:**
  - **instance**: Istanza singleton della classe.
  - **SAVE_FOLDER**: Directory per i file di salvataggio.
  - **LOG_FOLDER**: Directory per i file di log.
  - **currentLogFileName**: Nome del file di log corrente.
- **Metodi Principali:**

- **getNextGameNumber()**: Restituisce il numero progressivo per la partita.
- **salvaGioco(Gioco gioco, String fileName)**: Serializza l'oggetto **Gioco** e lo salva su file.
- **caricaGioco(String fileName)**: Carica lo stato del gioco deserializzando l'oggetto **Gioco** da file.
- **writeLog(String data)**: Scrive messaggi nel file di log con timestamp.
- **setCurrentLogFileName(String logFileName)**: Imposta il nome del file di log corrente.
- **renameLogFile(String oldFileName, String newFileName)**: Rinomina il file di log.

2.2.5 – Package **Utils**

Classe OutputUtils:

- **Descrizione:** Contiene metodi statici per la formattazione e la stampa del testo sulla console.
- **Attributi Principali:**
  - Costanti per i codici ANSI per i colori e gli stili del testo.
- **Metodi Principali:**
  - **print(String message, String... ansiCodes)**: Stampa un messaggio con i codici ANSI specificati.
  - **println(String message, String... ansiCodes)**: Stampa un messaggio con i codici ANSI specificati e va a capo.
  - **printTurnHeader(Giocatore giocatore)**: Stampa l'intestazione del turno per un giocatore.

Capitolo 3 – Funzionamento dell'Applicazione

3.1 – Flusso di Gioco

1. **Avvio dell'Applicazione:**
   - Viene mostrato un menu iniziale che permette di avviare una nuova partita o caricare una partita esistente.
2. **Creazione della Partita:**
   - Se viene scelta una nuova partita, i giocatori inseriscono i propri nomi e scelgono il colore delle armate.
   - Viene determinato l'ordine di gioco tramite il lancio dei dadi.
3. **Inizializzazione del Gioco:**
   - La mappa viene inizializzata con i continenti e i territori.
   - I territori vengono distribuiti casualmente tra i giocatori.
   - Ogni giocatore riceve un obiettivo segreto.
4. **Distribuzione Iniziale delle Armate:**
   - I giocatori distribuiscono le proprie armate iniziali sui territori controllati.

5. **Fasi di Gioco:**
    - **Distribuzione Armate:** All'inizio di ogni turno, i giocatori ricevono nuove armate in base al numero di territori e continenti controllati.
    - **Attacco:** I giocatori possono attaccare i territori avversari adiacenti per conquistarli.
    - **Spostamento Armate:** I giocatori possono spostare le proprie armate tra territori controllati adiacenti.
    - **Gestione Carte:** I giocatori possono scambiare combinazioni di carte per ottenere armate aggiuntive.
6. **Verifica Vittoria:**
    - Dopo ogni turno, viene verificato se un giocatore ha raggiunto il proprio obiettivo.
7. **Salvataggio e Caricamento:**
    - I giocatori possono salvare la partita in qualsiasi momento e riprenderla successivamente.
8. **Fine del Gioco:**
    - Quando un giocatore raggiunge il proprio obiettivo, il gioco termina e viene dichiarato il vincitore.

3.2 – Dettagli Implementativi

- **Gestione del Turno:**
    - Utilizzo della classe **TurnoGioco** per mantenere lo stato corrente del turno.
    - Le fasi del turno sono gestite tramite un **switch** sulla fase corrente.
- **Attacco e Difesa:**
    - Simulazione del lancio dei dadi tramite generazione di numeri casuali.
    - Confronto dei risultati dei dadi per determinare le perdite di armate.
- **Scambio di Carte:**
    - Implementazione delle regole per lo scambio di carte e calcolo delle armate ottenute.
- **Persistenza:**
    - Utilizzo della serializzazione Java per salvare e caricare lo stato del gioco.
    - Gestione dei file di salvataggio e dei log tramite la classe **FileServiceImpl**.
- **Interfaccia Utente:**
    - Utilizzo della libreria **Jansi** per migliorare l'esperienza utente con colori e formattazione del testo.
    - Gestione degli input dell'utente tramite la classe I**nputManagerSingleton.**.