

WILD TIME – DOCUMENTAZIONE

ESAME: METODI AVANZATI DI PROGRAMMAZIONE

PROGETTO A CURA DI: ROCCO PIAZZOLLA

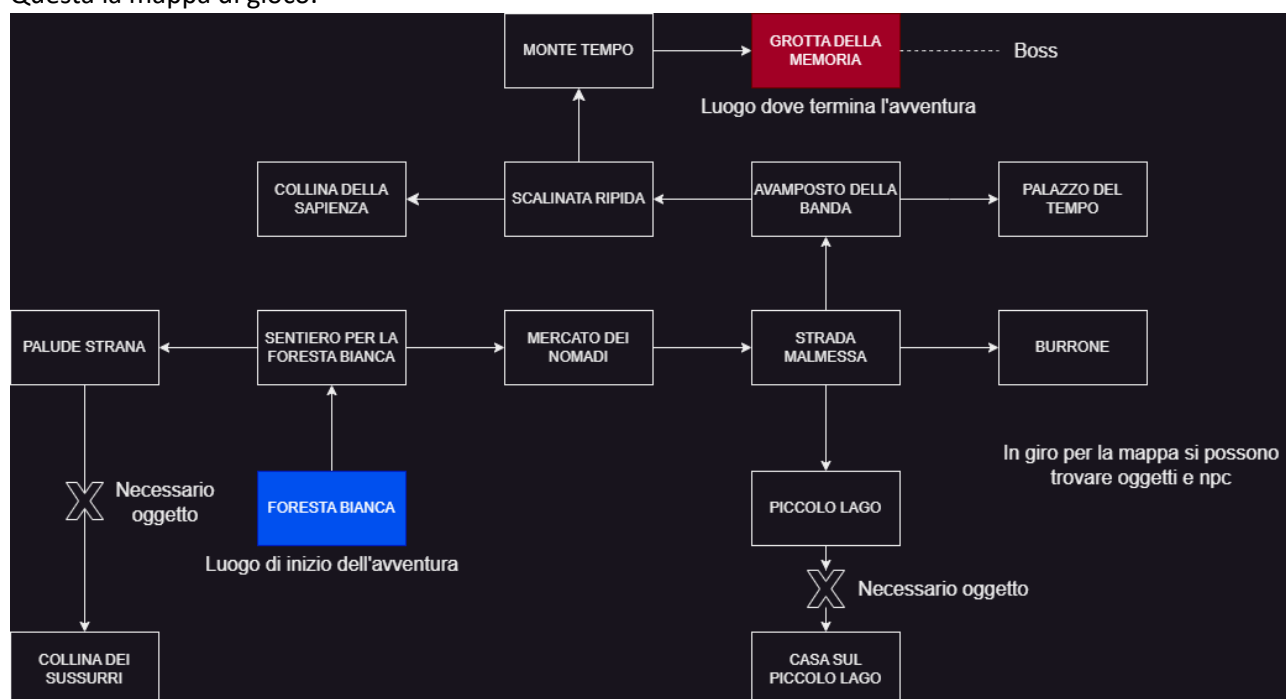
1. Introduzione

Per il corso di studi Metodi Avanzati di Programmazione è stato portato a termine il progetto per la realizzazione di un motore di gioco per avventure testuali. Per la sua realizzazione è stato utilizzato Java come linguaggio di programmazione.

2. Descrizione dell'avventura

Questa avventura ha luogo nelle immaginarie terre di Ooo, terre misteriose in cui strani eventi si verificano. Sei un mercenario assoldato da un re preoccupato che la banda che governa queste terre, la banda del Tempo, comandata dal temutissimo Rob Lucci, si impossessi dell'ingranaggio del tempo, un misterioso congegno che si trova in cima al monte più alto di tutte le terre, il Monte Tempo, nella Grotta della Memoria. L'ingranaggio per qualche strano motivo è in grado di controllare lo scorrere del tempo in queste lande e se venisse rubato tutto si fermerebbe e donerebbe chi lo possiede il potere di controllare qualsiasi cosa dato che è in grado di controllare il tempo. Il tuo compito è quindi quello di impedire a Lucci & Co che si impossessino dell'ingranaggio, che la caccia abbia inizio!

Questa la mappa di gioco.



3. Strutture dati

Per implementare le diverse funzionalità del motore, sono state definite le seguenti strutture dati:

- **Character**: definisce i personaggi dell'avventura
- **Npc**: Estensione della classe Character. Definisce i personaggi interagibili dall'utente.
- **Player**: estensione della classe Character. Definisce le caratteristiche del protagonista.
- **Command**: Definisce i comandi di gioco.
- **Inventory**: Definisce la composizione dell'inventario del personaggio principale.
- **Chest**: Definisce un contenitore di oggetti.
- **Item**: Definisce la composizione degli oggetti di gioco.

- **Room:** Definisce le caratteristiche delle stanze
- **Conversation:** Definisce la domanda che si può fare al npc e la risposta corrispondente.

Character

Variabile	Tipo	Valore Default	Significato
Hp	Int		Valore numerico punti vita del personaggio
Name	String		Nome del personaggio
Arma	Item	Null	Arma equipaggiata
Scudo	Item	Null	Scudo equipaggiato

Npc (estensione di Character)

Variabile	Tipo	Valore Default	Significato
isEnemy	Boolean	False	Definisce se il personaggio è un nemico
isTalking	Boolean	False	Definisce se il personaggio sta parlando
Reward	Item	Null	Definisce l'oggetto tenuto da un personaggio
Description	String		Definisce la descrizione del personaggio
Talk	String		Definisce la prima stringa di dialogo del personaggio
Conversation	List<Conversation>		Definisce la lista di domande e risposte che è possibile porre al personaggio

Player (estensione di Character)

Variabile	Tipo	Valore Default	Significato
MaxHP	Int	50	Definisce la salute massima del giocatore
Inventario	Inventory	New	Istanza di inventario associata al giocatore
Current_place	Room	Null	Definisce la stanza corrente in cui si trova il giocatore

Command

Variabile	Tipo	Valore Default	Significato
-----------	------	----------------	-------------

Name	String		Definisce il nome del comando
Alias	Set		Definisce un set di nome alternativi da poter utilizzare per richiamare il comando
Type	CommandType		Definisce il tipo di comando

Inventory

Variabile	Tipo	Valore Default	Significato
Items	List	New	Lista degli oggetti presenti nell'inventario

Chest

Variabile	Tipo	Valore Default	Significato
Chest	List	New	Definisce la lista di oggetti presenti nella cassa
isOpen	Boolean	False	Definisce lo stato della cassa (aperto / chiuso)
openWith	Item		Definisce l'oggetto tramite cui è possibile aprire la cassa

Item

Variabile	Tipo	Valore Default	Significato
Id	Int		Identificatore univoco dell'oggetto
Name	String		Nome dell'oggetto
Takeable	Boolean	False	Definisce se un oggetto si può prendere
Useable	Boolean	False	Definisce se un oggetto è usabile dal giocatore
isHeal	Boolean	False	Definisce se un oggetto è di tipo curativo
isWeapon	Boolean	False	Definisce se un oggetto è un'arma

isPowerUp	Boolean	False	Definisce se un oggetto è un potenziamento
Bonus	Int	0	Definisce il bonus dato al giocatore
attackDamage	Int	0	Definisce il danno d'attacco di un oggetto. Se 0 è uno scudo.
DefenseBonus	Int	0	Definisce la difesa di un oggetto. Se 0 è un'arma d'attacco.
Description	String		Definisce la descrizione di un oggetto

Room

Variable	Tipo	Valore Default	Significato
Id	Int		Identificatore dell'oggetto
Name	String		Definisce il nome dell'oggetto
Nord	Room	Null	Associa un'altra stanza a nord
South	Room	Null	Associa un'altra stanza a sud
Est	Room	Null	Associa un'altra stanza a est
West	Room	Null	Associa un'altra stanza a ovest
Chest	Chest	New	Definisce un contenitore di oggetti per la stanza
Items	List	New	Definisce la lista di oggetti presenti nella stanza
Npcs	List	New	Definisce la lista di npc presenti nella stanza
Description	String		Definisce la descrizione della stanza
isExplored	Boolean	False	Stabilisce se una stanza è stata esplorata
isBlocked	Boolean	False	Stabilisce se una stanza è bloccata
openWith	Item	Null	Definisce lo strumento tramite cui

			è possibile sbloccare la stanza
isFinal	Boolean	False	Definisce se la stanza è la stanza finale del gioco

Conversation

Variabile	Tipo	Valore Default	Significato
Question	String		Definisce la domanda da porre al personaggio
Answer	String		Definisce la risposta del personaggio alla domanda

4. Specifiche algebriche

Specifica algebrica della classe Room

Tipi: string, int, Room, List, Chest, Item

Operazioni:

getName() -> string

setName(string) -> Room

getId() -> int

setId(int) -> Room

getEst() -> room

setEst(Room) -> Room

getWest() -> Room

setWest(Room) -> Room

getNord() -> Room

setNord(Room) -> Room

getSud() -> Room

setSud(Room) -> Room

getDescription() -> String

setDescription(String) -> Room

getExplored() -> Boolean

setExplored(Boolean) -> Room

getBlocked() -> Boolean

setBlocked(boolean) -> Room

getFinal() -> Boolean

setFinal(boolean) -> Room

getItems() -> List

setItems(List) -> Room

getOpenWith() -> Item

setOpenWith(Item) -> Room

5. OO design

Classi principali

- **Engine:**

La classe Engine rappresenta il motore principale del gioco ed è responsabile dell'inizializzazione e dell'esecuzione del gioco stesso. Questa classe funge da punto di ingresso principale per l'applicazione e coordina il funzionamento generale del gioco. Al momento della creazione di un'istanza di Engine, si specifica il gioco da eseguire (in questo caso, WildTime) e quindi si chiama il metodo `execute()` per avviare il gioco.

- **GameManager:**

La classe GameManager è responsabile della gestione del ciclo di gioco, coordinando l'interazione tra il giocatore, l'interfaccia utente grafica (GUIManager), il Parser e le meccaniche di gioco. Questa classe gestisce l'avvio di nuovi giochi, il caricamento di giochi esistenti, l'elaborazione dei comandi del giocatore e la gestione di eventi cruciali come la morte del personaggio o la vittoria.

- **GUIManager:**

La classe GUIManager gestisce l'interfaccia utente grafica del gioco, fornendo metodi per aprire, aggiornare e chiudere le finestre di gioco e di inizializzazione delle stesse. È utilizzata per controllare l'interazione dell'utente con l'interfaccia grafica durante il gioco e per gestire l'apertura, l'aggiornamento e la chiusura delle finestre. Da questa classe si possono gestire due tipi di finestre, la finestra iniziale (StartFrame) e la finestra di gioco (GameFrame).

- **GameDescription:**

La classe astratta GameDescription rappresenta la descrizione generale di un gioco, in questo caso, di un'avventura testuale. Questa classe fornisce una struttura di base per definire le caratteristiche principali del gioco, inclusi comandi, stanze, personaggi non giocanti (NPC), oggetti e il giocatore. È progettata per essere estesa da classi specifiche che definiscono giochi con dettagli e comportamenti unici. La classe WildTime è un'estensione di questa classe.

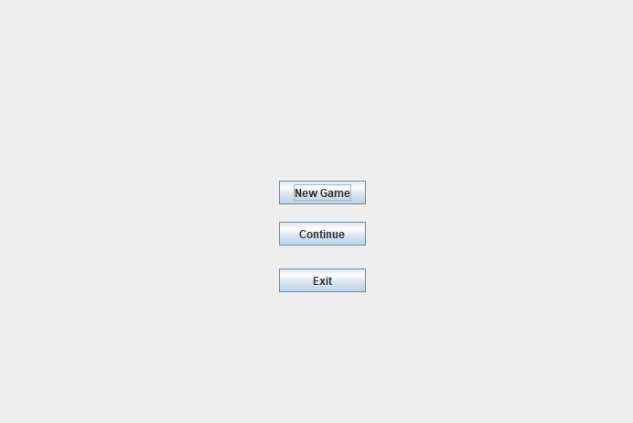
- **Parser:**

La classe Parser gestisce l'analisi dei comandi inseriti dal giocatore durante il gioco. Utilizza una serie di funzioni e filtri per identificare le parole chiave nella stringa di input, determinando la loro corrispondenza con gli oggetti e i comandi presenti nel gioco. Il metodo principale di questa classe è la funzione `parse` che ha come parametri un oggetto GameDescription e la stringa di testo inserita dall'utente e restituisce un oggetto ParserOutput contenente l'oggetto, il comando o l'npc corrispondente.

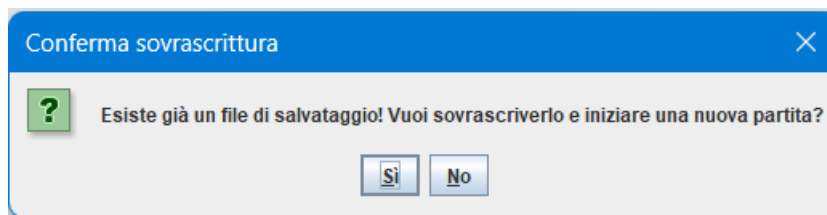
- **Utils**

La classe Utils fornisce diverse utilità per il caricamento e il salvataggio del gioco, la gestione delle azioni del giocatore e la verifica delle condizioni di vittoria o perdita. Queste utilità sono fondamentali per il corretto funzionamento e la fruizione del gioco da parte del giocatore.

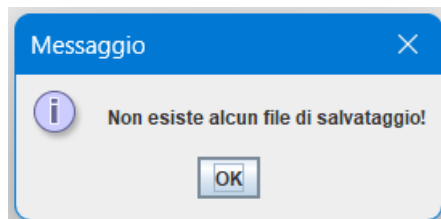
Di seguito il diagramma UML delle classi:



Nel caso in cui esista già un file di salvataggio, all'utente verrà chiesto se vuole sovrascrivere i dati creando un nuovo file di salvataggio.

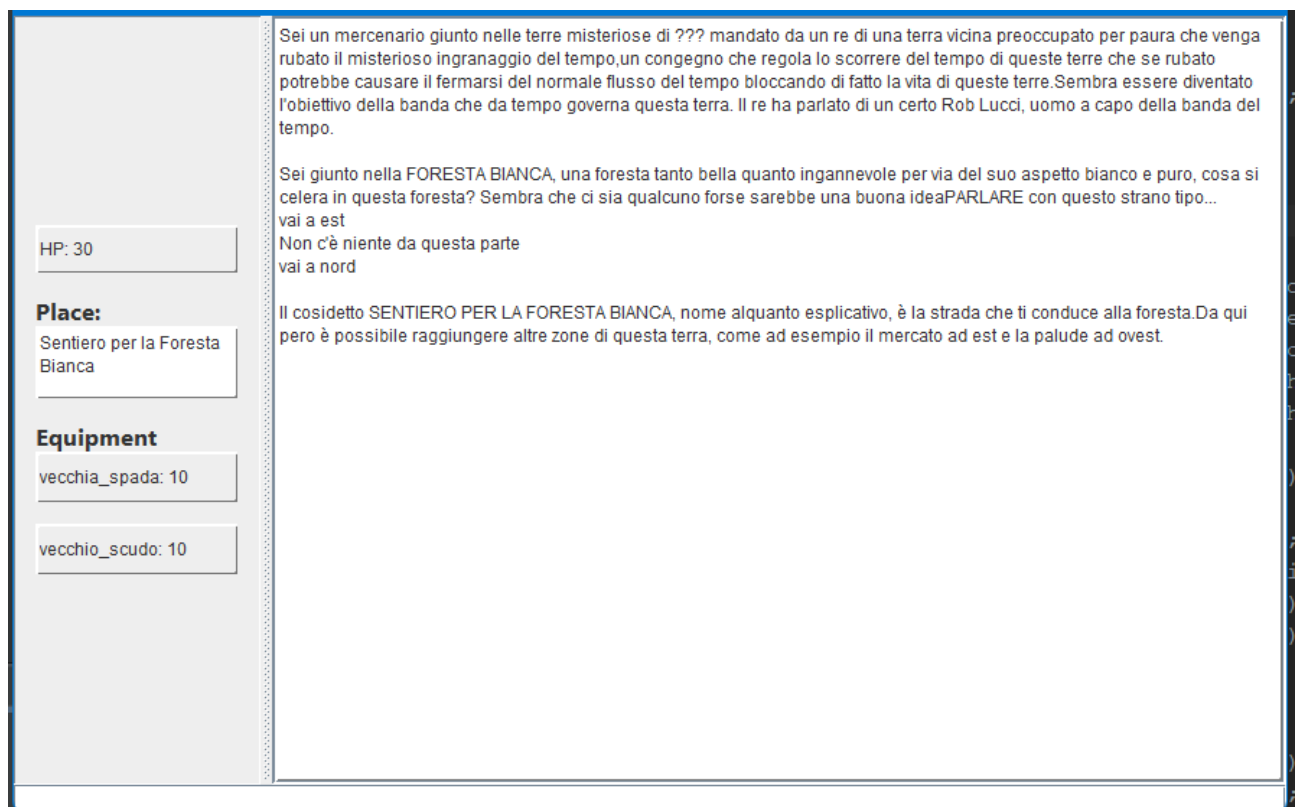


Nel caso in cui l'utente preme il tasto continua ma non esiste alcun file di salvataggio, verrà mostrato il seguente messaggio di errore



Nel caso in cui l'utente decida di iniziare una nuova partita, comincerà una nuova partita.

L'engine è provvisto di un'interfaccia di gioco. Nell'immagine sottostante uno screen di gioco. si distinguono tre aree: l'area dove l'utente potrà inserire il testo, la "Story Area" a destra e la "info Area" a sinistra. *Inserisci immagine*



La story area è quella porzione (lato destro dello SplitPanel) della finestra di gioco dove viene mostrato tutto ciò che avviene all'interno di gioco: spostamenti, comandi, interazioni e quant'altro.

La info area è l'area dedicata alle statistiche del giocatore (lato sinistro dello SplitPanel): punti vita, luogo attuale, arma equipaggiata e relativa potenza d'attacco, scudo equipaggiato e relativa difesa.

L'utente, nel caso in cui avesse bisogno d'aiuto, potrà vedere i comandi digitando "comandi" o "aiuto" e gli apparirà una lista di tutti i comandi disponibili.

Comandi di gioco:	
→ salva	→ per salvare la partita
→ esci	→ per uscire dalla partita e chiudere il gioco
→ pulisci	→ per ripulire l'area di testo
→ nord	→ per andare a nord della stanza
→ sud	→ per andare a sud della stanza
→ est	→ per andare a est della stanza
→ ovest	→ per andare a ovest della stanza
→ zaino	→ per vedere cosa hai nello zaino
→ apri (oggetto) (oggetto)	→ per aprire le casse
→ raccogli (oggetto)	→ per raccogliere un oggetto
→ parla (npc)	→ per parlare con un npc
→ osserva	→ per vedere cosa c'è nella stanza
→ mangia (o bevi)	→ per utilizzare consumabili
→ equipaggia	→ per equipaggiare armi o scudi
→ attacca (npc)	→ per attaccare un nemico
→ butta (oggetto)	→ per gettare via un oggetto presente nello zaino
→ usa (oggetto)	→ per usare gli oggetti utili ad aprire una stanza

È supportato anche l'utilizzo degli articoli.

Il gioco terminerà quando l'utente raggiungerà la stanza finale e avrà sconfitto il boss. Nel caso in cui il giocatore dovesse morire verrà riportato nella stanza iniziale del gioco con il 30% della vita massima disponibile.

7. Strumenti utilizzati

Sono stati utilizzati le Java Swing, Lambda Expressions, File e Thread.

Java Swing

Le java swing sono state implementate tramite la creazione di una classe GUIManager che gestisce i due tipi di finestra di cui dispone. StartFrame è la prima finestra che l'utente vede (vedi Manuale Utente) in cui sono presenti tre bottoni "New Game", "Continue", "Exit", il quale, una volta premuti, svolgono una funzione diversa: new game avvia una nuova partita mentre continue riprende la partita dall'ultimo salvataggio se disponibile, aprendo una nuova finestra quella del GameFrame.

Nel GameFrame sono state utilizzate diverse etichette per rappresentare le informazioni relative al giocatore, un textArea non editabile all'interno del quale viene mostrato tutto ciò che avviene all'interno del mondo di gioco, un textField provvisto di ActionListener in cui l'utente inserisce il testo che poi verrà processato, infine il GameFrame si compone di uno SplitPanel il quale permette la divisione dell'area superiore al textField in due parti.

File

I File sono stati implementati nel progetto per poter avere la possibilità di salvataggio dello stato di gioco. In questo modo è possibile interrompere una partita e riprenderla in qualsiasi momento.

Il File che viene chiamato è un file di tipo ".dat", e all'interno di esso sono contenute tutte le informazioni relative al gioco.

```

public static String LoadGame(GameDescription game, File saveFile) {
    String text = "";

    try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(saveFile))) {
        game = (GameDescription) inputStream.readObject();
        GameManager.setGame(game);
        text = "Gioco caricato con successo!\n";
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    return text;
}

```

Funzione che si occupa del caricamento del gioco da file.

Thread

I thread sono stati aggiunti per poter ritardare la chiusura immediata del gioco nel momento in cui l'utente decida di uscire dalla partita mediante il comando "esci" oppure termini la partita sconfiggendo il boss finale.

```

private static void PutThreadToSleep() {
    EventQueue.invokeLater(() -> {
        try {
            Thread.sleep(1300);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        guiManager.EndsGame();
    });
}

```

La funzione, presente nel GameManager, che si occupa di interrompere il thread corrente per 1,3 secondi prima di chiamare il metodo EndsGame del GUIManager che si occupa di rilasciare le risorse relative alle due finestre di gioco.