DOCUMENTAZIONE CASO DI STUDIO
PER IL CORSO DI METODI AVANZATI di
PROGRAMMAZIONE

DIPARTIMENTO
DI INFORMATICA

- Mezzina Michele, matricola: 697121
- Antonio Serino, matricola: 697401

INDICE

2. ASPETTI TECNICI

- Architettura del sistema
- Diagramma delle classi
- Specifica algebrica
- Dettagli di Implementazione
- Istruzioni per il primo utilizzo

---

1) L'AVVENTURA TESTUALE

• *SPUNTI E TRAMA DEL GIOCO*

Nonostante il titolo sembri essere quello di un film di Liam Neeson, l'avventura prende spunto dalla serie tv originale Netflix DARK: nella tranquilla cittadina di Windem succedono cose strane con una cadenza di 33 anni.
Il tutto è collegato a un disastro nucleare e a viaggi nel tempo.
Nonostante la serie tv divaghi parecchio abbiamo deciso di basare l'avventura su ciò che avviene all'interno della centrale nucleare.

Il nostro personaggio, Jhonas Khanwald, è uno dei pochi sopravvissuti al disastro nucleare, che oltre a causare l'apocalisse causa l'apertura di un varco spazio-tempo il quale permette di effettuare un viaggio nel passato, verso il momento e il luogo che più si desidera.

Mentre la stragrande maggioranza dei sopravvissuti reputa piacevole la vita post apocalittica (senza tasse, restrizioni e leggi di alcun tipo), il desiderio più grande del nostro protagonista è quello di poter evitare il disastro nucleare, aggiustando così la linea temporale.

Eccoci, dunque, all'inizio della nostra avventura: il nostro protagonista si troverà catapultato all'interno della centrale nucleare 5 minuti prima che il disastro avvenga.

ALLERTA SPOILER: È FORTEMENTE SCONSIGLIATO PER CHI NON HA ANCORA COMPLETATO IL GIOCO LEGGERE DA QUI IN POI, IN QUANTO VERRA' SPOILERATA LA SOLUZIONE DELL'AVVENTURA.

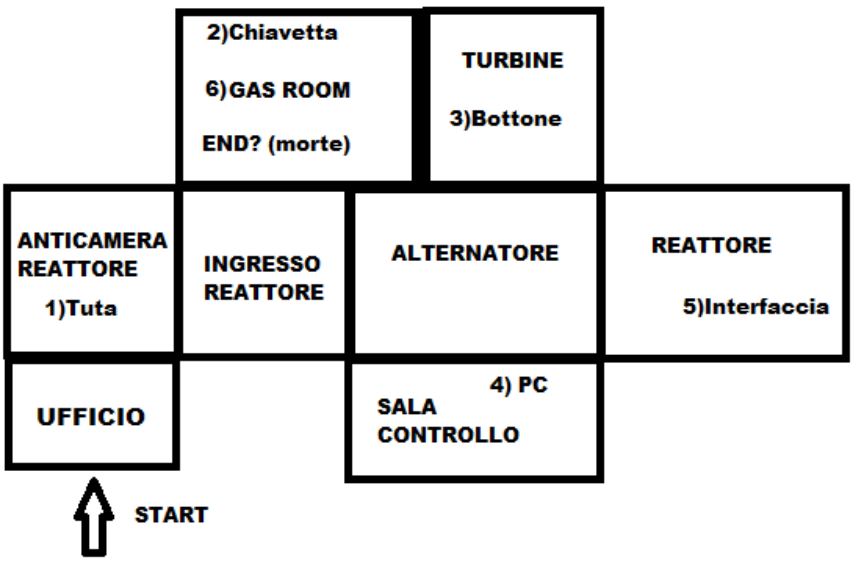
[Obiettivo e risoluzione]

Spostandosi tra le stanze della centrale nucleare (mappa), il nostro Jhonas, avrà l'obiettivo di scoprire il codice per disattivare l'esplosione del REATTORE H prima che il tempo (5 min) scada.

Il codice è possibile riceverlo utilizzando una *chiavetta* (recuperabile dalla STANZA DEGLI ESPERIMENTI, non prima di aver riattivato l'impianto di areazione per aspirare il gas nocivo all'interno). Successivamente, la chiavetta dovrà essere inserita all'interno del *PC* nella SALA CONTROLLO.

Naturalmente l'avventura del nostro Jhonas sarà resa complicata dalla ricerca delle giuste informazioni nelle stanze dell'intera mappa in cui, ad ogni passo falso, ci si può imbattere in pericoli di morte che potrebbero far terminare l'avventura nel peggiore dei modi.

- **<u>MAPPA</u>**



Come possiamo notare dalla mappa all'interno di ogni stanza sono presenti degli oggetti che il protagonista può utilizzare o aggiungere all'inventario. *Ad esempio, per poter accedere all'ingresso del reattore è indispensabile raccogliere e indossare la tuta che si trova nell'anticamera.*

- **COME SI GIOCA**

Al giocatore è permesso potersi spostare lungo gli ambienti della mappa con i quattro punti cardinali: nord, sud, est e ovest.
In qualsiasi stanza il personaggio si trovi verrà fornita di default una descrizione dell'ambiente e la possibilità di poter utilizzare il comando "guarda" che darà indicazioni sull'eventuale presenza di oggetti nella stanza e informazioni geografiche per la mossa successiva.

- **LISTA DEI COMANDI**

Per quanto riguarda la lista dei comandi dobbiamo distinguere i comandi di movimento dai comandi d'azione.

Quelli di **movimento** sono i seguenti:

nord	Nord-NORD
sud	Sud-SUD
est	Est-EST
ovest	Ovest-OVEST

Quelli d’**azione** invece:

inventario	inv-inventario
osserva	guarda-vedi-trova-cerca-descrivi
raccogli	prendi
usa	utilizza-applica-indossa-inserisci
accendi	attiva
premi	spingi
fine	end-esci-muori-ammazzati-ucciditi-suicidati-exit

Inventario: fornisce una lista degli oggetti raccolti.

Osserva: permette di avere indicazioni su cosa c’è nella stanza e informazioni riguardanti l’eventuale movimento del protagonista.

Raccogli: permette di aggiungere oggetti al proprio inventario.

Usa: permette di usare oggetti presenti nell’inventario.

Accendi: permette di accendere eventuali oggetti, rendendoli utilizzabili. (PC)

Premi: permette di premere eventuali oggetti premibili. (BOTTONE)

Fine: permette di terminare la partita in corso.

(NELLE IMMAGINI RIPORTATE A SINISTRA TROVIAMO IL NOME COMPLETO DEL COMANDO E A DESTRA ALIAS UTILIZZABILI).

---

2) ASPETTI TECNICI

- ## ARCHITETTURA DEL SISTEMA

L'avventura, scritta interamente in Java, presenta al proprio interno un'architettura tale da permetterci di lavorare in modo organizzato e idoneo al raggiungimento degli obiettivi prefissati.

La classe principale per il funzionamento dell'avventura è la classe **Engine**: questa permette l'esecuzione di giochi che estendono la classe GameDescription.

La classe **GameDescription** è una classe astratta (in quanto possiede due metodi init() e nextmove() astratti) la cui finalità è quella di poter essere riutilizzabile: chi creerà un'avventura avrà l'obbligo di implementarla basandosi su GameDescription.

Nel nostro caso la classe **SerinoAdventures** sarà l'estensione di GameDescription(quindi andrà ad effettuare l'overriding sui metodi init() e nextmove() ed aggiunge ulteriori metodi per la gestione di database e dettagli di gioco): al suo interno si trova l'intera avventura che verrà passata all'engine.

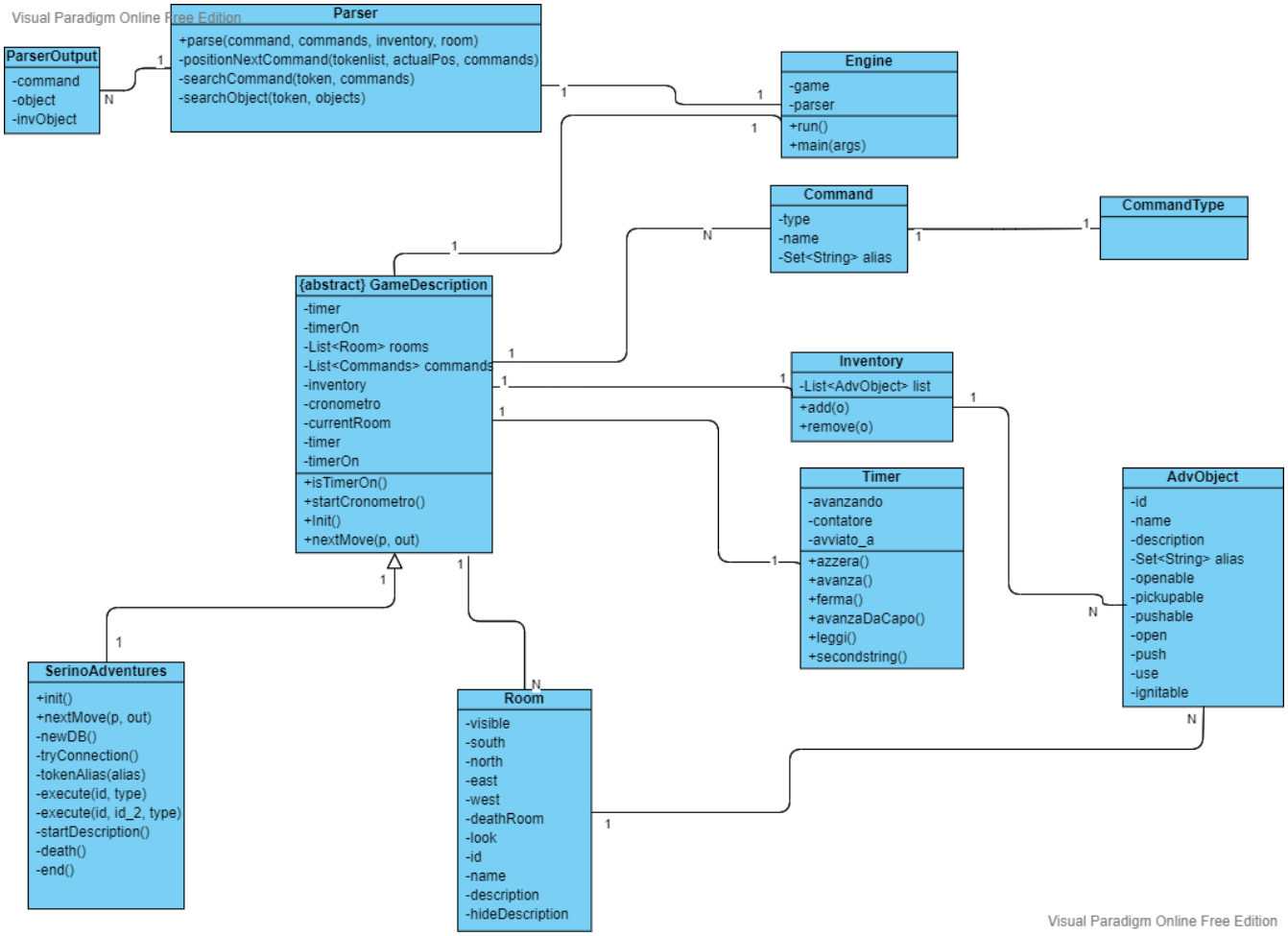
L'interpretazione delle stringhe di testo in comandi è garantita dalla classe **Parser**: fornita una frase questa creerà un token per ogni parola interrotta da uno spazio, riconoscerà tra le parole tokenizzate quali sono comandi e quali sono oggetti e infine restituirà una coda di oggetti di tipo **Parseroutput**, ovvero delle descrizioni dei comandi. Ogni elemento della coda verrà eseguito dal metodo nextmove() in base al modo in cui l'avventura creata decide di implementarla. In particolare, in SerinoAdventure, è importante specificare che nextmove() si avvarrà di due metodi execute() (con un relativo *overloading*): il primo dei due ci permette di poter effettuare azioni *(es. usa, premi, etc.)* con un solo oggetto *(es. "usa la tuta")* mentre il secondo ci permetterà di effettuare azioni specificando due oggetti, uno dall'inventario e uno presente nella stanza (es. "inserisci chiavetta nel PC).

Un utile supporto al gioco è stato apportato con l'utilizzo di un **DBMS H2** per poter memorizzare tutti i dati riguardanti oggetti, comande e stanze.

Il DB ci permette di poter modificare dinamicamente i dati (oggetti, comande e stanze) in SerinoAdventure.

Sono presenti, inoltre, altre classi come **Room**, **Command**, **Inventory**, **Timer**, **AdvObject**, al cui interno troviamo proprietà e metodi dei relativi agli elementi base di cui si compone un'avventura.

- **DIAGRAMMA DELLE CLASSI**



- **SPECIFICA ALGEBRICA**

CODA

Una coda è un tipo di dato dinamico in quanto viene creata inizialmente vuota (mediante un operatore **emptyqueue**), poi gli elementi vengono aggiunti uno alla volta mediante una operazione di inserimento (operatore

enqueue). Gli elementi possono essere eliminati dalla coda uno alla volta con una operazione di eliminazione (operatore **dequeue**). È possibile verificare se una coda è vuota mediante un operatore booleano **codavuota**. È possibile leggere il valore della prima variabile inserita, quella più vecchia cronologicamente, senza modificarlo mediante l'operatore **top**.

SPECIFICA SINTATTICA:

TIPI: coda, boolean, elemento

OPERATORI:

- **Emptyqueue**: crea una coda vuota
- **Enqueue**: aggiunge elementi a una coda
- **Dequeue**: elimina un elemento dalla coda
- **Top**: legge il valore del primo elemento nella coda
- **Codavuota**: operatore booleano che ci dice se la coda sia piena o vuota

OPERAZIONI:

Emptyqueue()→coda

Enqueue(coda,elemento)→coda

Dequeue(coda)→coda

Top(coda)→elemento

Codavuota(coda)→boolean

COSTRUTTORI E OSSERVAZIONI:

osservazioni	**costruttori**	
	emptyqueue()	enqueue(coda,elemento)
codavuota(coda')	true	false
top(coda')	error	**If** codavuota(coda) **then** elemento **else** top(coda)
dequeue(coda')	error	**If** codavuota(coda) **then** emptyqueue **else** enqueue(dequeue(coda),elemento)

SPECIFICA SEMANTICA:

considerando una coda c, un elemento e ed un valore booleano(true/false):

codavuota(emptyqueue)	true
codavuota(enqueue(c,e))	false
dequeue(enqueue(c,e))	**if** codavuota(c) **then** emptyqueue **else** enqueue(dequeue(c),e)
top(enqueue(c,e))	**if** codavuota(c) **then** e **else** top(c)

- ## DETTAGLI DI IMPLEMENTAZIONE

Avventure di qualità

Ogni avventura ha un gameplay migliorato grazie alle potenzialità del parser implementato.

All'interno del package "parser" è presente la classe "Parser.java" che, collaborando con la classe "ParserOutput.java", permette l'elaborazione

delle stringhe testuali acquisite durante una sessione di gioco. Il parser analizza la stringa in input, tokenizza ogni parola su ogni spazio bianco e crea una coda di token.

In particolare, se una parola corrisponde ad un comando/oggetto questa avvalora un'apposita variabile, altrimenti viene considerata come "stop word" quindi viene automaticamente scartata. Questa operazione permette all'utente di inserire comandi del tipo: "Vai a NORD poi vai ad EST" in cui le parole: "Vai", "a", "poi", "vai", "ad" vengono escluse mentre "NORD" ed "EST" vengono riconosciuti come comandi.

Successivamente, ogni qual volta vengono individuati dei comandi dalla lista (token), il parser, cerca nei token successivi degli oggetti correlati permettendo di unirne fino a due oggetti ad uno stesso comando (un oggetto nell'inventario e uno nella stanza) per permettere all'utente operazioni del tipo: "Usa chiavetta all'interno del pc" in cui "usa" è il comando e "chiavetta" e "pc" sono gli oggetti.

Per ogni comando individuato, viene generato un oggetto di tipo "ParserOutput" appositamente configurato con eventuali legami tra oggetti e comandi.

Infine, il parser restituisce una coda di mosse da compiere corrispondenti alla frase inserita dall'utente.

Avventure a tempo

Nel package "type" è stato aggiunto un nuovo componente fondamentale per le avventure temporizzate, la classe "Timer.java".

Un oggetto di tipo "Timer" utilizza dei metodi, basati sulla sincronizzazione dei thread (synchronized, vincola l'esecuzione del metodo ad un solo thread per volta), per calcolare il tempo di esecuzione intercorso dall'avvio del gioco.

Nonostante la sua implementazione, temporizzare le avventure non è obbligatorio, difatti se l'"Engine" riconosce che un'avventura non ha avvalorato un oggetto di tipo "Timer", questo non imporrà limiti di tempo di gioco.

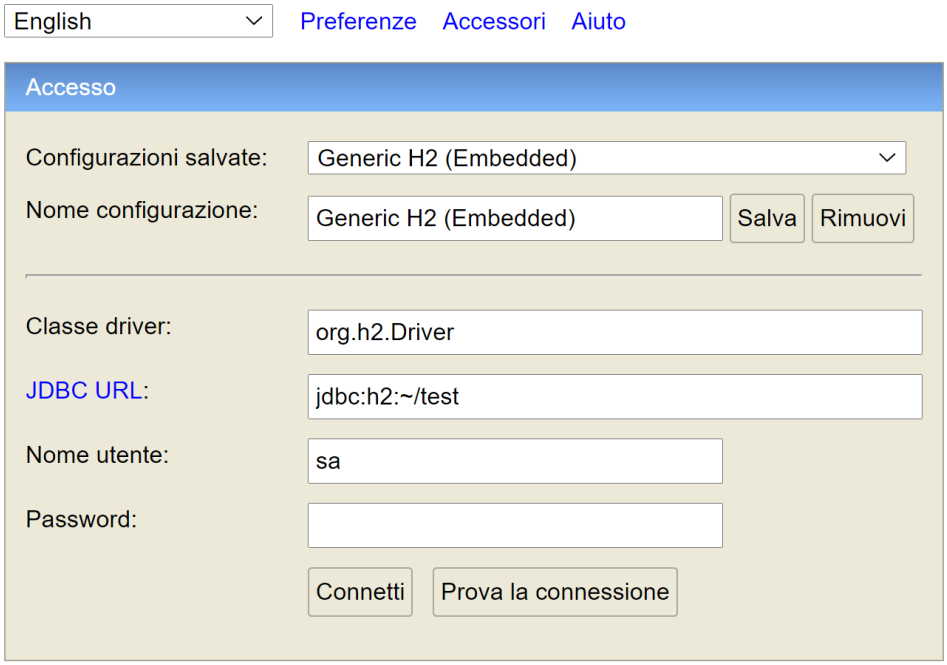
Database, per aggiornamenti dinamici

All'interno dell'avventura testuale le descrizioni inerenti ai comandi, gli oggetti e le stanze sono caricati dinamicamente da un database, in questo modo il progettista può aggiornare con più semplicità i dati di gioco evitando di accedere direttamente al codice sorgente. L'implementazione dei database è affidata ad "H2", un gestore di basi di dati relazionale, di cui è disponibile anche una console con cui è possibile interagire direttamente sulle tabelle del database.

Per i giocatori invece è stata predisposta una funzione apposita che configura un database locale con i dati di gioco, qualora questo non sia presente.

- **<u>ISTRUZIONI PER IL PRIMO UTILIZZO</u>**

Prima di avviare il gioco per la prima volta è prudente installare la ***console H2*** che permette di verificare la connessione al DB locale



Configurare i vari parametri come in figura (se non è mai stata utilizzata la console H2 dovrebbero apparire già correttamente configurati)
Successivamente, avviando il gioco (assicurandosi di non essere collegati contemporaneamente al DB) dovrebbero generarsi automaticamente delle tabelle pre-configurate dell'avventura.

Infine, godersi l'avventura.