HyperV

Indice

1. Descrizione del caso di studio

HyperV è un’avventura testuale sviluppata in Java, ambientata in un laboratorio. L'obiettivo del gioco è recuperare un potente virus, che potrebbe essere utilizzato per diversi scopi. Il giocatore deve esaminare attentamente l'ambiente circostante, prestando particolare attenzione ai dettagli e prendendo decisioni strategiche.

Per progredire nel gioco è essenziale usare l’ingegno e le capacità analitiche per interpretare gli indizi nascosti nelle descrizioni delle stanze e degli oggetti e per risolvere i complessi enigmi. Il giocatore potrà così sbloccare nuove aree, fino a raggiungere la stanza finale in cui è custodito il virus. Ogni passo avanti richiede un'attenta riflessione e la capacità di collegare i vari indizi sparsi lungo il percorso, rendendo l'esperienza di gioco immersiva e stimolante.

2. Istruzioni di gioco

2.1 Avvio del programma

Per avviare il programma, è necessario seguire due passaggi fondamentali:

1. Avviare il server

Eseguire il file *Engine*, che crea e gestisce il server necessario per la comunicazione client-server; questa operazione consente di stabilire una connessione e scambiare dati tra il client e il server.

2. Avviare il client

Eseguire il file *HyperVGui*, che rappresenta l'interfaccia grafica del gioco; il client si connetterà al server precedentemente avviato e consentirà all'utente di giocare all'avventura testuale.

2.2 Comandi di gioco

Comandi di navigazione tra le stanze:

- nord (oppure N): permette di spostarsi a nord;
- sud (oppure S): permette di spostarsi a sud;
- est (oppure E): permette di spostarsi a est;
- ovest (oppure O): permette di spostarsi a ovest.

Comandi di interazione con gli oggetti:

- prendi [nome oggetto]: raccoglie un oggetto presente nella stanza e lo aggiunge all'inventario;
- sblocca [nome oggetto] [password]: sblocca un oggetto utilizzando una password;
- usa [nome oggetto]: utilizza un oggetto.

Comandi generali:

- inventario (oppure INV): mostra gli oggetti che possiedi;
- osserva: descrive la stanza in cui ti trovi e gli oggetti al suo interno;
- help: mostra una breve descrizione dell'obiettivo del gioco, seguita dall'elenco dei comandi disponibili, correlatidi descrizione.

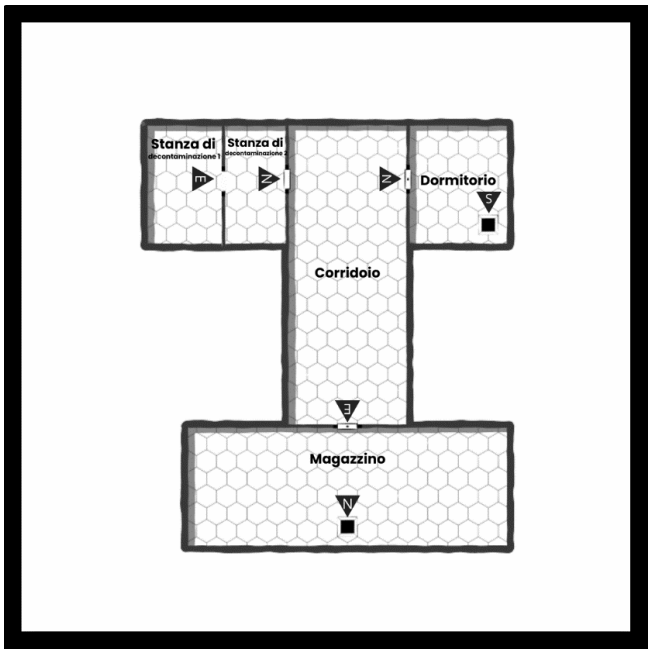
Comandi per terminare il gioco:

- muori: termina il gioco perché vieni ucciso da un robot presente nel laboratorio;
- rosso/provetta: crei un nuovo composto in grado di controllare le menti, vinci il gioco;
- blu/becher: provochi un'esplosione che distrugge il laboratorio, perdi il gioco.

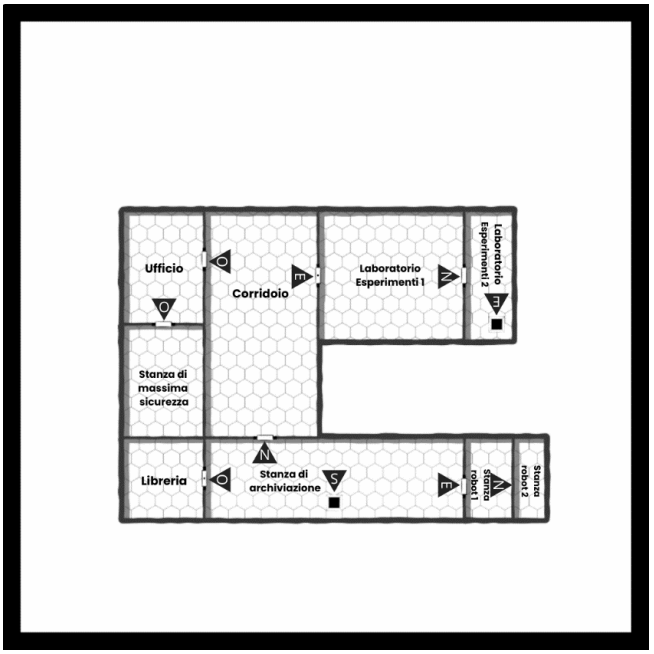
2.3 Mappa di gioco

Il gioco si articola su tre piani, le cui stanze sono tutte visitabili.

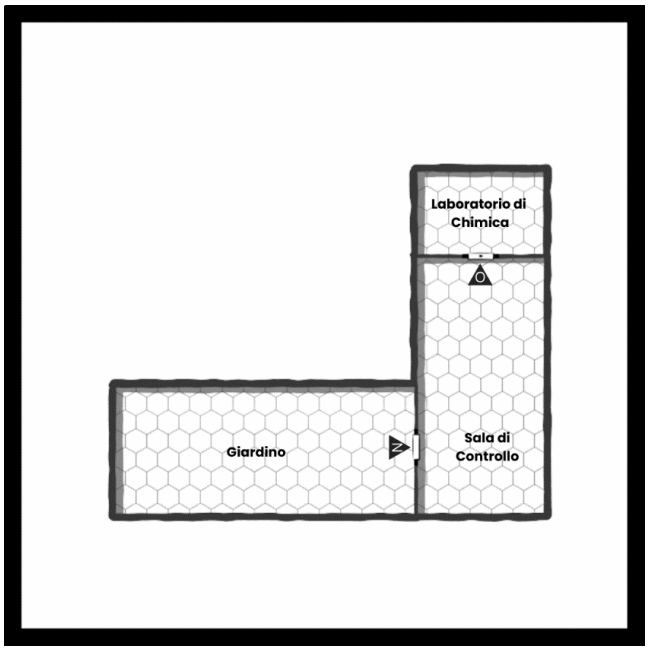
La mappa è visibile anche all'interno del gioco e mostra la posizione del giocatore.



Piano terra



Primo piano

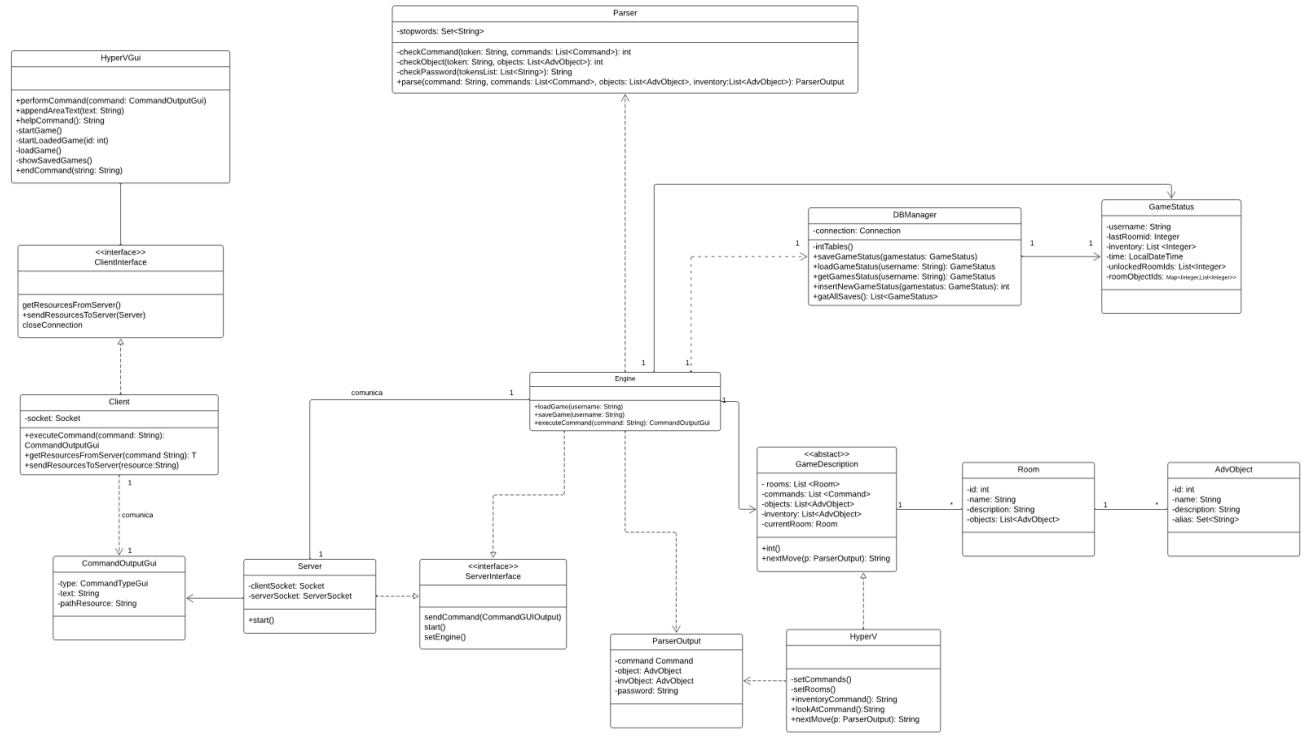


Secondo piano

2.4 Soluzione ottimale del gioco

- E
- Prendi Chiave
- N
- Usa Chiave
- E, N, E
- Prendi Badge
- S, N
- Usa Badge
- E, N, N
- Prendi Torcia
- S, E, S
- Usa Torcia
- O, N
- Prendi Portal Gun
- Risolvi enigma (soluzione furto)
- Sblocca Porta furto
- S, O, S, E
- Usa Portal Gun
- N
- Prendi Laser
- S, S
- Usa Laser
- O
- Prendi Nota
- Usa Nota
- Usa Portal Gun
- N
- Prendi becher, prendi provetta
- N
- Sblocca porta virus
- O
- Rosso

3. Diagramma delle classi



Descrizione diagramma delle classi

• *Engine*, gestisce l'esecuzione complessiva del gioco. Questo componente centrale si occupa di avviare il gioco, mantenerne lo stato e coordinare l'interazione tra le varie parti del sistema. L'Engine si interfaccia con il Parser per interpretare i comandi del giocatore, con il GameDescription per accedere alle informazioni sul mondo di gioco e con il DBManager per la gestione dei dati persistenti.

• *Parser*, responsabile di analizzare i comandi inseriti dal giocatore, trasformandoli in una struttura comprensibile per l'Engine attraversi algoritmi di parsing e tokenizzazione.

• *DBManager*, gestisce la connessione al database e fornisce funzionalità per inizializzare le tabelle necessarie, ottenere lo stato di gioco di un utente, inserire nuovi stati di gioco e recuperare quelli salvati. Comunica con l'Engine per fornire le informazioni sullo stato di gioco richieste e per memorizzare i nuovi stati di gioco.

• *GameStatus*, tiene traccia di informazioni essenziali come il nome utente, la posizione corrente e l'inventario.

• *GameDescription*, è una classe astratta che rappresenta la descrizione completa del gioco, inclusi i dettagli sullestanze, i comandi disponibili, l'inventario e gli oggetti nel gioco.

Viene estesa dalla classe *HyperV* e inizializzata dall'Engine all'avvio del gioco, implementa la logica specifica del gioco, definendo comandi, stanze e meccaniche di interazione.

• *Room,* rappresenta una stanza nel gioco, con gli attributi relativi e i metodi per accedere agli stessi.

• *AdvObject,* rappresenta un oggetto nel gioco, con gli attributi relativi e i metodi per accedere agli stessi.

• *Client e Server,* gestiscono la comunicazione in rete permettendo il gioco in un ambiente locale e il salvataggio.

• *ClientInterface* e *ServerInteface,* definiscono i protocolli di comunicazione tra il client e il server, garantendo una corretta interazione tra le parti.

• *HyperVGui,* che implementa l'interfaccia grafica utilizzando la libreria JSwing, cura l'aspetto visivo del gioco. Fornisce i componenti grafici necessari per l'interazione con il giocatore. Comunica con l'Engine per inviare comandi e ricevere output visibili all'utente.

• *CommandOutputGui,* si occupa di gestire la rappresentazione visuale dell'output dei comandi, fungendo da intermediario tra il Client e l'interfaccia grafica principale del gioco.

4. Specifica algebrica

Si prende in esame la struttura dati Lista della classe Inventory (con riferimento alla classe AdvObject), diconseguenza il metodo set è già inizializzato.

SPECIFICA SINTATTICA

Sorts: List, Item, Integer, Boolean

Operations

newList() -> List	Crea una lista vuota
addList(List, Item, Integer) -> List	Aggiunge un elemento alla lista nella posizione specificata
isEmpty(List) -> Boolean	Restituisce true se la lista è vuota altrimenti false
getSize(List) -> Integer	Restituisce il numero di elementi contenuti nella lista
getIndex(List, Item) -> Integer	Restituisce la posizione dell'elemento specificato
getItem(List, Integer) -> Item	Restituisce l'elemento nella posizione specificata
remove(List, Integer) -> List	Rimuove dalla lista l'elemento nella posizione specificata
contains(List, Item) -> Boolean	Restituisce true se l'elemento specificato è contenuto nella lista

Item: è un tipo generico usato come placeholder. Può essere rimpiazzato da qualunque tipo di dato.
Integer: numeri interi
Boolean: valori di verità (vero o falso)
Integer e Boolean sono tipi ausiliari alla definizione della specifica algebrica della lista

OSSERVAZIONI E COSTRUTTORI

Costruttori di l’		
Osservazioni	newList	add(l, it, id)
isEmpty(l')	true	false
getSize(l')	0	if isEmpty(l) then 1 else getSize(l) + 1
getIndex(l', it')	error	if it = it' then id else getIndex(l, it')
getItem(l', id')	error	if id = id' then it else getItem(l, id')
remove(l', id')	error	f id = id' then l else add(remove(l, id'), it)
contains(l', it')	false	if it = it' then true else contains(l, it')

SPECIFICA SEMANTICA

DECLARE

- l, l': List
- it, it': Item
- id, id': Integer

OPERATIONS

- isEmpty(newList) = true
- isEmpty(add(l, it, id)) = false
- getSize(newList()) = 0
- getSize(add(l, it, id)) = if isEmpty(l) then 1 else getSize(l) + 1
- getIndex(add(l, it, id), it') = if it = it' then id else getIndex(l, it')
- getItem(add(l, it, id), id') = if id = id' then it else getItem(l, id')
- remove(add(l, it, id), id') = if id = id' then l else add(remove(l, id'), it)
- contains(newList, it') = false
- contains(add(l, it, id), it') = if it = it' then true else contains(l, it')

SPECIFICA DI RESTRIZIONE

RESTRICTIONS

- getIndex(newList, it') = error
- getItem(newList, id') = error
- remove(newList, id') = error

dove ‘error’ è un elemento speciale indefinito

5. Applicazione argomenti del corso

• **File**: utilizzati per caricare descrizioni che forniscono dettagli sull'ambiente circostante e gli oggetti presenti con cuiè possibile interagire, oltre ai nomi delle stanze e le stopwords.

• **JBDC**: utilizzato per la gestione dei salvataggi. Permette di salvare lo stato di gioco e tutte le informazionirelative.

• **Thread**: si utilizza un thread per gestire il timer, inserito all'interno di uno degli enigmi presenti nel gioco. Implementato all'interno della classe *WordleGame,* il thread viene impiegato per creare un'operazione asincrona che funziona in parallelo al flusso principale del gioco. Questo approccio consente di separare la logica del conteggio del tempo dalle altre operazioni del gioco, migliorando la reattività e l'efficienza complessiva dell'applicazione.

• **Socket**: utilizzati per implementare un'architettura client-server. La comunicazione tra server e client è infatti resa possibile attraverso socket, che gestiscono lo scambio di dati e comandi.

• **SWING**: utilizzato per creare un'interfaccia grafica interattiva che consente all'utente di interagire con il gioco, rendendo l'esperienza più coinvolgente.

• **Lambda expression**: utilizzate per rendere il codice più leggibile, efficiente e conciso rispetto all'utilizzo di classiinterne o anonime e per ridurre la quantità di codice di supporto necessario.

Nel nostro caso, vengono ad esempio utilizzate in metodi relativi agli action listener della GUI o nell'inserimento degli oggetti nelle stanze, in seguito al caricamento di una partita salvata. L'uso della lambda expression in questo contesto permette di cercare efficacemente un oggetto in una lista basandosi sul suo ID, senza dover scrivere un ciclo esplicito.

• **Interfacce**: utilizzate per garantire una possibile scalabilità del modo in cui avviene la comunicazione Client-Server. Le interfacce ClientInterface e ServerInterface forniscono un'astrazione delle **funzionalità** essenziali richieste rispettivamente dal client e dal server, promuovendo una chiara separazione delle responsabilità e facilitando la manutenzione del codice.

L'interfaccia *ClientInterface* delinea i metodi necessari per la comunicazione del client con il server, includendo l'invio di comandi, la richiesta di risorse e la gestione della connessione. Implementata dalla classe Client, questa interfaccia garantisce una struttura coerente per le operazioni lato client, permettendo potenziali implementazioni alternative senza modificare il contratto di base.

Parallelamente, l'interfaccia *ServerInterface* definisce le operazioni fondamentali del server, come l'avvio del servizio, l'invio di comandi al client e l'interazione con il motore di gioco. La classe Server, che implementa questa interfaccia, incapsula la logica di gestione delle connessioni e di elaborazione delle richieste dei client.

L'utilizzo di queste interfacce nel progetto favorisce la modularità e la flessibilità del sistema.

• **Generics**: utilizzati per scrivere codice che può lavorare con diversi tipi di dati, mantenendo allo stesso tempo la sicurezza dei tipi a compile-time. L'uso dei generics nel contesto Client-Server permette al metodo di essere flessibile, potendo restituire diversi tipi di oggetti a seconda di ciò che il server invia, senza perdere la sicurezza dei tipi.