DOCUMENTAZIONE CASO DI STUDIO

- Antonio Silvestre, matricola 697697
- Edoardo Pomarico, matricola 697698

INDICE:

1. **Avventura grafica:**
   - 1.1 Introduzione e trama del gioco
   - 1.2 Mappa
   - 1.3 Regole del gioco
2. **Documentazione tecnica:**
   - 2.1 Architettura del sistema
   - 2.2 Diagramma delle classi
   - 2.3 Dettagli di implementazione
   - 2.4 Specifica algebrica

1) Avventura grafica:

1.1) Introduzione e trama del gioco

Lo stile di gioco utilizzato è un'avventura testuale e avventura grafica. Abbiamo cercato di prendere da entrambi gli stili qualcosa in modo da creare un'avventura "diversa" che non rientrasse necessariamente nei canoni degli stili citati prima.
L'avventura è svolta in prima persona e l'interazione tra il gioco e l'utente è gestita esclusivamente da dei tasti da cliccare con il mouse all'interno dell'interfaccia.
Il gioco ha una trama apparentemente banale: lo scopo del personaggio principale è quello di convalidare il voto dell'ultimo esame. Si nasconde però una velata critica all'intero sistema universitario (nomi e situazioni sono frutto della nostra fantasia).
Sono presenti anche degli indovinelli e non è detto che il giocatore riesca a terminare il gioco in quanto c'è una disponibilità limitata di monete (utili per sbloccare determinate stanze e per proseguire).
Sapendo che non tutti possono essere pratici in questo tipo di giochi, abbiamo pensato di aggiungere una voce "Aiuto", disponibile solamente all'inizio del gioco, con qualche piccolo consiglio utile. Alla fine dell'avventura sarà assegnato un punteggio che dipenderà dal tempo impiegato per concluderla ed è disponibile una tabella in cui si possono vedere i risultati degli altri giocatori.

1.2) Mappa del gioco

Essendo il gioco ambientato in un luogo con più piani (in una università) abbiamo deciso di implementare nell'avventura una "mappa dinamica" che in base alla stanza corrente stamperà a schermo ciò che c'è nei vari punti cardinali. Per attivarla (o disattivarla) basta premere la voce "Gioco" e successivamente la voce "Mappa".

1.3) Regole del gioco

Al protagonista è permesso potersi spostare tra i vari ambienti, cliccando sui tasti direzionali (N, S, E, O, ecc…).

SPOILER ALLERT!

Il gioco potrebbe concludersi prima del previsto nel caso in cui il giocatore prenda tre caffè che non siano lunghi.

2) Documentazione tecnica

2.1) Architettura del sistema

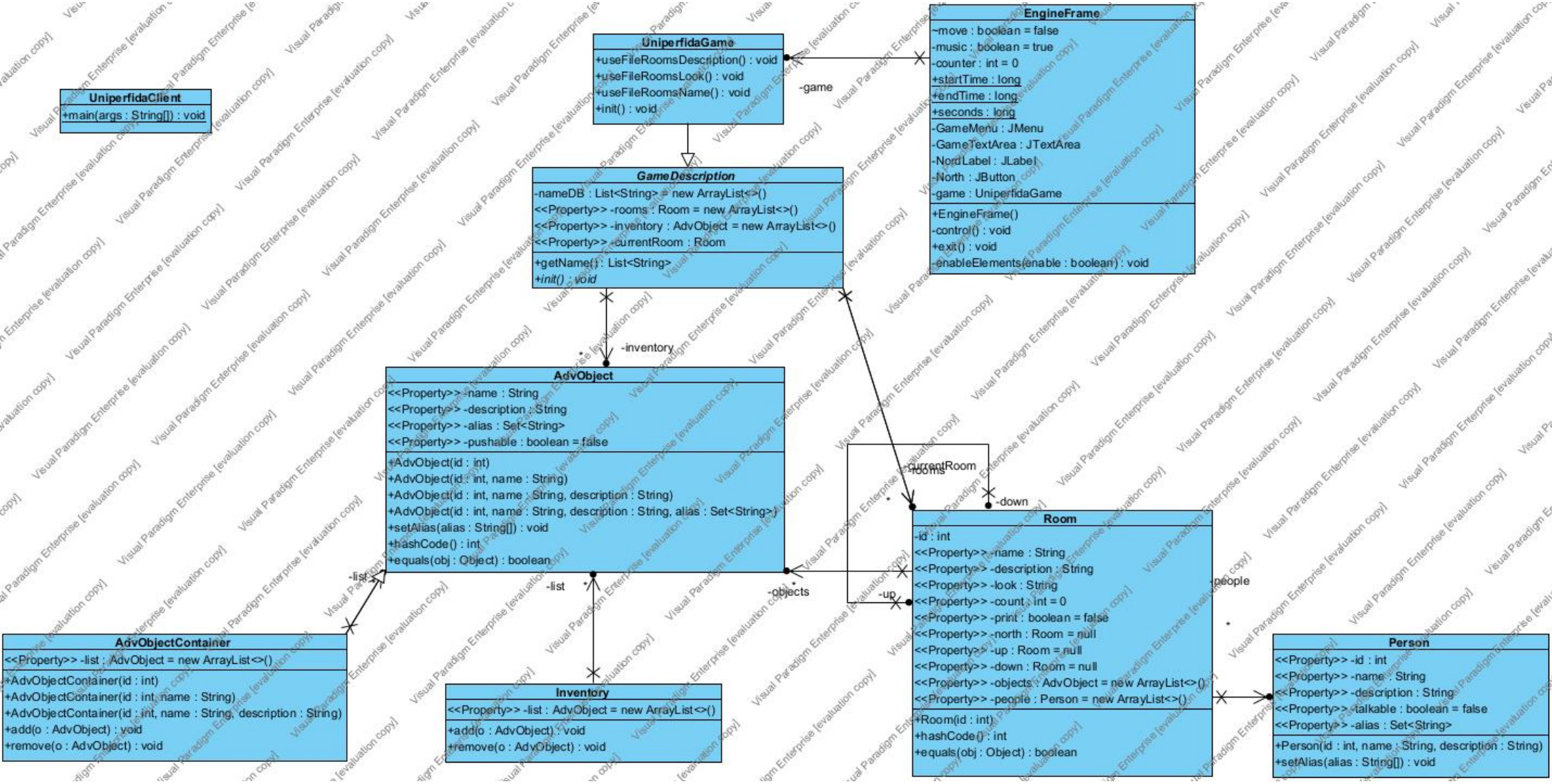
L'avventura, realizzata in Java, presenta un'architettura di sistema suddivisa in tre parti fondamentali (package):

1. Frame: contiene tutto ciò che riguardi l'interfaccia grafica.
2. Games: contiene la definizione di tutti gli oggetti del gioco.
3. Type: contiene tutti i tipi di oggetti utili al gioco ovvero:

- AdvObject (tutti gli oggetti del gioco come ad esempio le monete)
- AdbObjectContainer (tutti gli oggetti contenitore come ad esempio la macchinetta del caffè)
- Inventory (l'inventario che nel nostro caso è utilizzato come un borsellino capace di contenere solamente monete)
- Person (tutte le persone con cui è possibile interagire)
- Room (tutte le stanze)

È presente inoltre un ulteriore pacchetto chiamato "unused" all'interno del quale sono presenti delle classi non più utilizzate.

2.2) Diagramma delle classi



2.3) Dettagli di implementazione

L’avventura sfrutta le potenzialità degli elementi di “javax.swing”. Questo è il framework di Java che permette la realizzazione di interfacce grafiche (GUI).
Un utile supporto al gioco è stato apportato con l’utilizzo di un DBMS H2 per poter memorizzare i dati dei giocatori (nome, punteggio e data e ora in cui si è giocato)
Inoltre sono stati utilizzati (in maniera non propriamente corretta*) i Socket che permettono al giocatore (client) di connettersi ad un server in modo da avere un codice eseguibile molto leggero in quanto tutto il motore grafico del gioco è caricato sulla classe server.
I Socket potenzialmente possono essere utilizzati per garantire a più giocatori di giocare contemporaneamente.

* generalmente server e client non risiedono sulla stessa macchina ma il mio collega ed io abbiamo deciso di inserirli sulla stessa macchina perché non ci è stato possibile testare il gioco su due macchine diverse.

Dato che sono presenti due main all’interno del progetto, per poter giocare bisogna seguire questi passaggi (riportati anche nel file README presente all’interno della cartella del progetto):

- Spostare la cartella "uniperfidaProgettoMap" nel Desktop
- Aprire il cmd o PowerShell
- Digitare "cd Desktop"
- Digitare "cd uniperfidaProgettoMap"
- Digitare "cd uniperfida"
- Digitare "cd target"
- Digitare "java -cp Uniperfida-1.0-SNAPSHOT-jar-with-dependencies.jar di.uniba.map.b.uniperfida.frame.EngineFrame
(Dovrebbe comparire una scritta "In attesa di una connessione da parte di un client")
- Aprire una nuova scheda del cmd o di PowerShell (nel caso in cui non fosse possibile aprire cmd se si hai aperto PowerShell o aprire cmd se hai aperto PowerShell
- Digitare "cd Desktop"
- Digitare "cd uniperfidaProgettoMap"
- Digitare "cd uniperfida"
- Digitare "cd target"
- Digitare "java -cp Uniperfida-1.0-SNAPSHOT-jar-with-dependencies.jar di.uniba.map.b.uniperfida.frame.UniperfidaClient
- Giocare

(È necessario scaricare la versione 11 di Java)

2.4) Specifica algebrica e struttura dati

LISTA

Una lista è una sequenza finita di elementi dello stesso tipo. A ogni elemento viene assegnata una posizione (pos(i)) e un valore. Si può accedere direttamente al primo valore, mentre per accedere ad un elemento generico bisogna scandire quelli precedenti. Possiamo dire che è una struttura dati di tipo dinamico in quanto è possibile aggiungere (inserisciLista) e rimuovere (cancellaLista) elementi. Questo è possibile perché questa struttura è a dimensione variabile. Inoltre, è possibile verificare se una lista è vuota tramite un operatore booleano listaVuota e leggerne un qualsiasi elemento leggiLista.

SPECIFICA SINTATTICA:

TIPI: Lista, booleano, posizione, elemento

OPERATORI:

- creaLista: crea una lista vuota;
- listaVuota: verifica se una lista è vuota;
- leggiLista: legge un elemento della lista;
- scriviLista: scrive un elemento all'interno della lista;
- inserisciLista: aggiunge un altro campo alla lista;
- cancellaLista: rimuove un campo dalla lista.

OSSERVAZIONI:

creaLista() → Lista
listaVuota(Lista) → Booleano leggiLista(posizione,Lista) → Elemento
scriviLista(elemento,posizione,Lista) → Lista
inserisciLista(elemento,posizione,Lista)→Lista cancellaLista(posizione,Lista)→Lista

SPECIFICA ALGEBRICA:

Supponendo che L sia una lista, e sia un elemento, p una posizione e b un booleano:

OSSERVAZIONI		COSTRUTTORI
	creaLisa()	inserisciLista(e,p,L)
listaVuota(L')	true	false
leggiLista(p',L')	error	If (p'=p) then L else leggiLista(p',L')
cancellaLista(p',L')	error	If (p'=p) then L else inserisciLista (e,p,cancellaLista(p'L))
scriviLista(e',p',L')	error	inserisciLista(e',p',cancellaLista(p',L'))

SPECIFICA SEMANTICA:

Prendiamo in considerazione una lista L, un elemento e, ed un valore booleano true/ false:

listaVuota(creaLista)	true
listaVuota(scriviLista(e,p,L))	false
cancellaLista(p,scriviLista(e,p,L))	If listaVuota(L) then creaLista () else scriviLista(e,p,cancellaLista(p,L))
leggiLista(p,scriviLista(e,p,L))	If listaVuota(L) then e else leggiLista(p,L)

FINE