Università degli studi di Bari facoltà di scienze
MM.FF.NN

Progetto Metodi Avanzati di Programmazione

Phosphorus textual-adventure

by

Vito Proscia mat. 735975

Email: v.proscia3@studenti.uniba.it

Repository github: Phosphorus

Anno accadenico 2022-2023

Contents

1 Introduzione

1.1 Definizione obiettivo principale

L'obiettivo principale del progetto è la realizzazione di un'avventura testuale in Java, inglobando, in essa, le tecniche e gli argomenti trattati durante il corso di Metodi Avanzati di Programmazione.

1.2 Trama

Il protagonista, l'agente f24, si trova su di una navicella spaziale di ritorno alla Terra da una missione che ha consistito nel catturare alieni per produrre il fosforo necessario alla sopravvivenza del pianeta, infatti, sulla quest'ultimo, il fosforo, che riveste un ruolo fondamentale per la sopravvivenza dei vegetali e quindi per il sostentamento dell'uomo è cominciato a diminuire drasticamente, per questo si organizzano spedizioni per catturare alieni in grado di produrlo.

Inizialmente, f24 si sveglierà dal sonno criogenico nel dormitorio con un ordine, impartito dal comandante, di indagare sulla misteriosa scomparsa di due alieni prigionieri. Il protagonista cercherà i due fuggitivi, districandosi tra le stanze dell'astronave ed interrogando i membri dell'equipaggio, fino a scoprire cosa viene fatto agli alieni prigionieri. Sarà solo a lui decidere se mantenere lo *status quo* o ribellarsi.

1.3 Soluzione gioco

La stanza iniziale è il dormitorio, la prima cosa che si deve fare è prendere la bombola d'ossigeno che sarà utile più avanti, una volta parlato con l'agante a13 bisogna recarsi alla sala meeting, a nord del dormitorio, dopo aver ascoltato il comandante si dovrà prendere la pistola e ci si recherà ad ovest per la mensa, lì ci saranno due scienziati s56 e s99, per proseguire ci si dovrà spostare ad est verso la sala macchine, una volta dentro ci si imbatterà nel primo nemico, l'alieno Orionix, una volta sconfitto bisogneà prendere la chiave dello sgabuzzino, dopo di chè ci si dovrà entrare andando a sud, una volta all'interno si dovrà prendere il bigliettio che conterrà il codice per accedere al laboratorio (4815) che si troverà appena a nord della sala meeting, una volta entrati troveremo l'ultimo nemico Nebulor, che sarà protetto da un altro scienziato, s47, una volta parlato con quest'ultimo starà al giocatore decidere se eliminare l'ultimo nemico o prendere la modifica delle pistola per cercare di cambiare la situazione.

1.4 Mappa di gioco

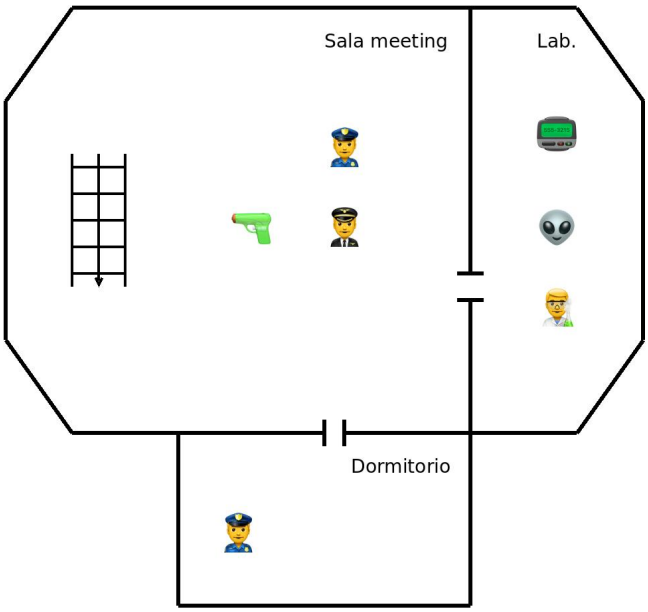

Figure 1: Mappa primo piano

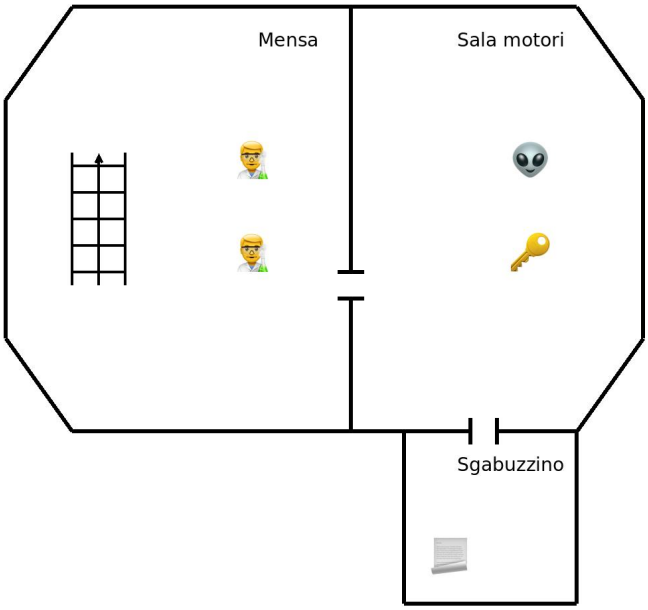

Figure 2: Mappa secondo piano

2 Dettagli implementativi

In questa sezione verranno analizzate tutte le tecniche e gli argomenti del corso sfruttati per la realizzazione dell'avventura testuale.

2.1 Progettazione Object-Oriented

La progettazione orientata agli oggetti (OOD) è un paradigma di programmazione che utilizza "oggetti" come unità fondamentali per progettare, organizzare e implementare software. Gli oggetti sono istanze di classi, che sono astrazioni che rappresentano concetti, dati e operazioni associati al dominio del problema che si sta affrontando, questa metodologia di progettazione garantisce riutilizzabilità del codice, facilità di manutenzione, chiarezza nel flusso di controllo e scalabilità.

Nello specifico si sono andate a progettare e realizzare una serie di classi garantendo l'incapsulamento di dati e operazioni, di modo che l'accesso a quest'ultimi sia limitato ai metodi della stessa classe, il che promuove la sicurezza e la modularità del codice.

Al fine di chiarire la struttura delle classi e delle loro relazioni, abbiamo utilizzato il linguaggio di modellazione UML (Unified Modeling Language) per rappresentare in modo visuale la progettazione del software in modo chiaro e conciso.

Il diagramma delle classi è presente al seguente link: class_diagram_uml

2.2 File

Nel progetto i file sono stati utilizzati per modellare gli elementi narrativi e non, dell'avventura, in particolare si hanno a disposizione tre file JSON (JavaScript Object Notation) che vanno a descrivere le stanze, i personaggi e gli oggetti di gioco in modo che le informazioni legate a questi siano facilmente modificabili ed estendibili per migliorare l'avventura, magari aggiungendo più stanze, modificando i dialoghi dei personaggi e perfino stravolgere la trama di gioco.

All'avvio del gioco avviene la lettura di questi file per inizializzare tutti i vari oggetti che costituiscono l'avventura, questo avviene utilizzando la libreria jackson, che permette di leggere e di utilizzare facilmente i file di tipo JSON, convertendoli in una mappa chiave-valore.

```
"0": {
        "roomId": 0,
        "roomName": "Dormitorio",
        "roomDescription": "Dormitorio non molto grande ...",
        "lookDescription": "Sei nel dormitorio, vicino a te c'è ...",
        "floorNumber": 0,
        "locked": false,
        "south": null,
        "north": 1,
        "west": null,
        "est": null,
        "items": [
            4
        ],
        "characters": [
            0
        ],
        "oxygen": true,
        "passwordRequired": false
    },
```


(a) File delle stanze


```
"0": {
        "characterId": 0,
        "characterName": "a13",
        "mainDialog": "Buongiorno f24, ho saputo che sei nuovo, ...",
        "defaultDialog": "Hai sentito il comandante, dobbiamo andare ...",
        "type": "friend"
    },
```


(b) File dei personaggi


```
"0": {
        "itemID": 0,
        "itemType": "Weapon",
        "itemName": "pistola",
        "itemDescription": "Particolare pistola, ...",
        "itemLocation": 1,
        "itemAction": "spara",
        "itemAlias": [
            "pistola",
            "arma"
        ]
    },
```


(c) File degli oggetti di gioco

2.3 Database

I salvataggi di gioco sono stati gestiti mediante l'uso delle basi di dati utilizzando JDBC (Java Database Connectivity), in particolare sono state progettate tre relazioni:

- game: memorizza i dati relativi al giocatore, come l'id, la stanza in cui si trova, il numero di nemici eliminati ed il timestamp del salvataggio;
- inventory: raccoglie tutti gli oggetti recuperati durante l'avventura;
- killedCharacter: conserva i dati relativi ai personaggi eliminati nel corso dell'avventura.

Ogni qual volta che viene salvata la partita, se si esegue il primo salvataggio viene creato il database con le sue relazioni per poi inserire i relativi dati, altrimenti con il nuovo salvataggio vengono sovrascritti i dati del vecchio.

Si è pensato di sovrascrivere sempre il salvataggio per motivi di semplicità, ma con l'attuale sistema è possibile modificare quest'aspetto dando la scelta al giocatore di salvare/caricare più sessioni.

Esempio:

GAMEID	CURRENTROOMID	ENEMYCOUNT	GAMETIME	SAVETIMESTAMP
0	4	1	34	2024-01-30 11:08:48.017

Figure 3: Relazione game

ITEMID	GAMEID
0	0
4	0

Figure 4: Relazione inventory

CHARACTERID	GAMEID
5	0

Figure 5: Relazione killedCharacter

2.4 Threads

I Thread sono unità di esecuzione meno complesse dei processi, essi consentono ad un programma di eseguire più operazioni in modo concorrente, migliorando l'efficienza e consentendo l'esecuzione simultanea di diverse attività.
Per il progetto i thread non stati usati per due scopi differenti:

1. `di.uniba.map.game.GameTimer`: per scandire il tempo di gioco;
2. `di.uniba.map.App`: per la riproduzione della musica in sottofondo durante l'esecuzione del gioco, per ulteriori dettagli sulla composizione e produzione della musica, consulta l'Appendice A.

2.5 Api REST

Le API RESTful (Representational State Transfer) sono un'architettura di comunicazione stateless basata su standard web che utilizza metodi HTTP standard, ogni risorsa è unica e raggiungibile attraverso URI (Uniform Resource Identifier). Nel nostro caso le API sono state utilizzate per recuperare le informazioni relative alla qualità dell'aria della città di Bari (41.12° N, 16.86° E) quali:

- European AQI: indice europeo della qualità dell'aria;
- UV index: livello di radiazione UV solare sulla superficie terrestre;
- Ozone O_3: quantità di ozono nella stratosfera;
- Sulphur dioxide SO_2: quantità del gas nocivo diossido di zolfo nell'ambiente;
- Nitrogen dioxide NO_2: quantità del gas nocivo diossido di azoto nell'ambiente;

Si è scelto di reprerire questi valori per una maggiore immersività nella'avventura. Esempio:

Indicatori ambient.	Valori	Esito
EU_AQI	28	FAIR
Ozone O_3	71.00 µg/m³	FAIR
Sulphur Dio. SO_2	1.30 µg/m³	GOOD
UV_index:	1.95	GOOD
Nitrogen Dio. NO_2	4.40 µg/m³	GOOD

Figure 6: Risultato della chiamata API con valutazione

2.6 Java Swing

Java Swing è un toolkit grafico incluso nella libreria standard di Java per la creazione di interfacce utente (UI) per applicazioni desktop.
Il framework è stato utilizzato per la creazione di un tastierino numerico con i tasti cliccabili (keypad) usato per accedere al laboratorio inserendo la giusta password.

Figure 7: Interfaccia grafica keypad

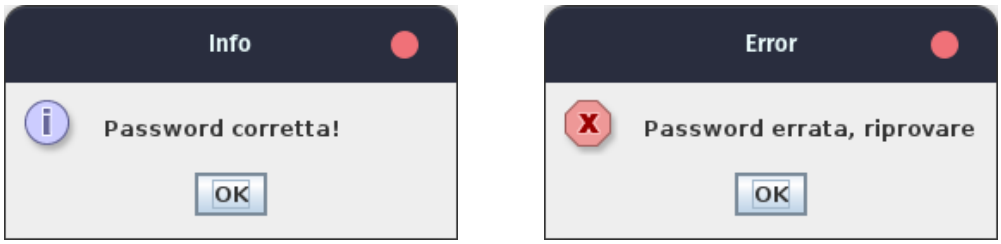

Figure 8: Messaggi in caso di risposta corretta/errata

3 Sommario classi implementate

In questa sezione sarà presentata una breve panoramica delle classi implementate nel progetto.

3.1 `di.uniba.map`

In questo package sono contenute la classi principali per l'avvio del gioco:

- App: classe che fa partire il gioco, contiene il metodo *main*;
- Utils: classe contenente tutti i vari metodi di supporto alle altre classi del progetto.

3.2 `di.uniba.map.game`

Nucleo centrale del progetto, questo package ingloba tutte le classi necessarie al funzionamento del gioco collegando le altre classi insieme:

- AirQuality: classe che si occupa di gestire la chiamata API al servizio di open-meteo;
- GameEngine: classe che regola lo scheletro di gioco inglobando tutte le varie azioni effettuabili, le stanze disponibili, l'inventario e la stanza corrente;
- GameTimer: si occupa dell'esecuzione del thread timer;
- PhosphorusGame: gestisce le varie azioni disponibili, sia del menù che del gioco vero e proprio;
- SaveGame: classe che coordina i salvataggi di gioco, contiene i metodi per salvare e riprendere la sessione di gioco.

3.3 `di.uniba.map.parser`

Le classi all'interno di questo package gestiscono l'input del giocatore convertendolo in azioni di gioco:

- Parser: classe che analizza l'input del giocatore cercando di meglio interpretalo per convertire il tutto in azioni di gioco (azione, azione-oggetto e azione-personaggio);
- ParserOutput: classe che compone il comando che il giocatre impartisce.

3.4 `di.uniba.map.type`

Questo package contiene tutti i tipi progettati che compongono il gioco:

- Action: rappresenta le azioni di gioco;
- ActionType: contiene tutte le tipologie di azioni;
- Character: descrive le caratteristiche dei personaggi di gioco;
- Enemy: specializzazione della classe Character che gestisce le informazioni dei nemici;
- Inventory: rappresenta l'inventario di gioco;
- Item: descrive un oggetto di gioco;
- KetItem: specializzazione di Item per rappresentare gli oggetti chiave;
- Room: rappresenta le stanze le compongono il gioco;
- Weapon: specializzazione della classe Item che rappresentare gli oggetti che possono danneggiare i personaggi.

3.5 `di.uniba.map.ui`

Le classi contenutevi compongono l'interfaccia utente del gioco:

- JKeypad: classe che gestisce la GUI (Graphical User Interface) del tastierino numerico utile ad accedere al laboratorio andando ad inserire una passowrd;
- UI: contiene tutti i metodi relativi alla stampa a schermo delle varie componenti del gioco, quali la mappa, il menù, i vari finali, etc....

4 Conclusioni

In conclusione, l'obiettivo principale del progetto è stato lo sviluppo di un'avventura testuale in Java, mettendo insieme tutte le tecniche studiate durate il corso M.A.P. con la possibilità di aggiungere creatività per legare il tutto.

Personalmente mi ritengo soddisfatto del risultato finale, sia in termini di progettazione che di realizzazione, il tutto mi ha permesso di migliorare assimilando il paradigma di programmazione ad oggetti, donandomi un nuova *forma mentis* che sicuramente mi aiuterà ad affrontare ed a risolvere problemi futuri.

5 Sviluppi futuri

Nonostante il raggiungimento degli obiettivi preposti il progetto è aperto a sviluppi futuri che possono migliorare l'esperienza dei giocatori, eccone alcuni esempi:

1. Implementazione di una vera e propria GUI;
2. Possibilità di creare/caricare più salvataggi;
3. Aggiunta di componenti queli il livello di salute del giocatore e dei nemici, con possibilità di far attaccare quest'ultimi.

Queste sono alcune delle idee da implementare per rendere il gioco più accattivante e divertente.

A Musica

In questa sezione appendice verranno presentati tutti gli step della creazione della colonna sonora di gioco "short_circuit" che ho realizzato per migliorare l'esperienza del giocatore, mettendo a frutto la mia grande passione per la musica.

A.1 Composizione

Per arricchire l'atmosfera ho incluso nel progetto una mia improvvisazione musicale, cercando di trasmettere un po' quel senso di mistero che avvolge la vicenda di gioco, al fine di far immergere il giocatore ancor meglio nell'avventura.

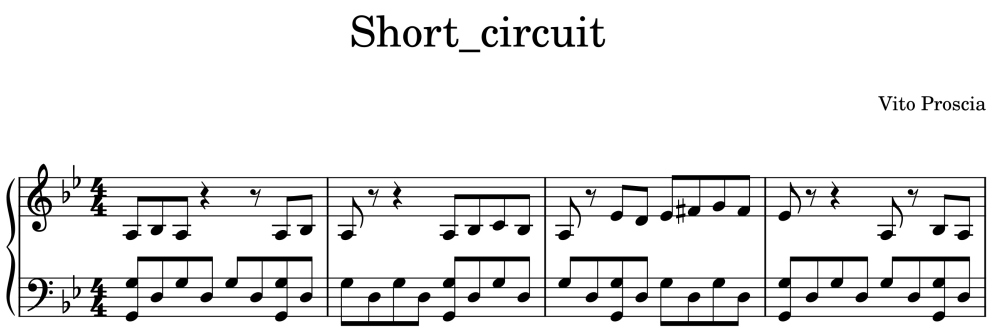

Figure 9: estratto spartito di "Short_circuit"

A.2 Produzione

La parte relativa alla produzione musicale è stata una delle più ardue ed al contempo una delle più divertenti, personalmente non mi ero mai cimentato, fino a questo momento, nella produzione di un brano musicale.

Nello specifico, con la funzione record del pianoforte digitale, ho registrato il brano per poi salvarlo in formato MIDI (Musical Instrument Digital Interface), successivamente ho esportato il file nell'applicazione per iPad Garageband, che permette, amatorialmente, di creare musica, per poi modificare lo strumento di esecuzione, aggiungere degli effetti sonori come il wabble ed il riverbero ed andare ad editare la traccia audio, tagliando o registrando nuovamente delle parti, infine ho salvato il tutto sotto froma di WAV (Waveform) per poi imporare il brano nel progetto facendo sì che sia riprodotto in loop durante il gioco.

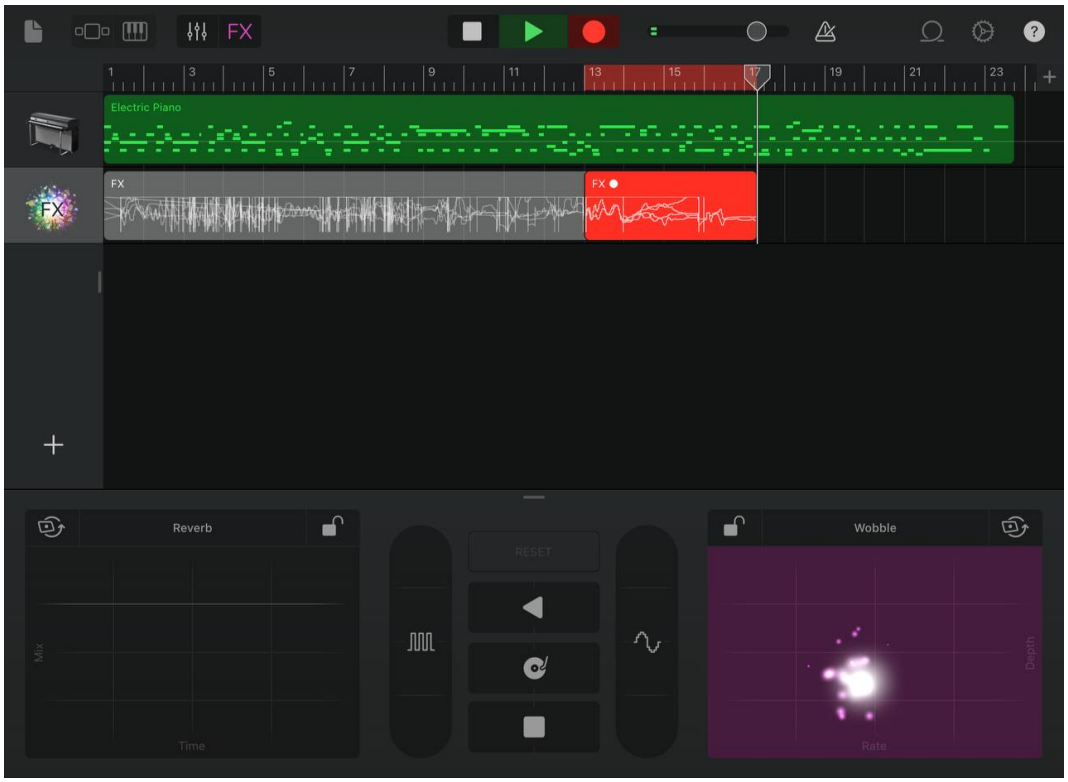

Figure 10: Screenshot dell'applicazione Garageband

B Specifica algebrica della Lista

In questa sezione appendice verrà presentata la specifica algebrica della struttura dati Lista.

B.1 Specifica sintattica

Sorts: Lista, Elemento, Boolean, Integer, Posizione.

Operations:

- creaLista() → Lista
- listaVuota(Lista) → Boolean
- leggiLista(Lista, Posizione) → Elemento
- cancLista(Lista, Posizione) → Lista
- insLista(Lista, Elemento) → Lista
- lunghezza(Lista) → Integer
- contiene(Lista, Elemento) → Boolean

B.2 Specifica semantica

Osservatori	Costruttori di l'	
	creaLista()	insLista(l, p, i)
listaVuota(l')	true	false
leggiLista(l', p')	error	$if\ p = p'\ then\ i\ else$ leggiLista(l, p')
cancLista(l', p')	error	$if\ p = p'\ then\ l\ else$ insLista(cancLista(l, p'), p, i)
lunghezza(l')	0	lunghezza(l) + 1
contiene(l', i')	false	$if\ i = i'\ then$ true $else$ contiene(l, i')

Table 1: Tabella per la costruzione degli operatori della specifica semantica

Declere:

- l : Lista<Elemento>

- i : Elemento
- p : Posizione
- $true, false$: Boolean

Operations:

- listaVuota(creaLista()) $\rightarrow$ true
- listaVuota(insLista(l, p, i)) $\rightarrow$ false
- leggiLista(insLista(l, p, i), p') $\rightarrow$ $if\ p = p'\ then\ i\ else$ leggiLista(l, p')
- cancLista(insLista(l, p, i), p') $\rightarrow$ $if\ p = p'\ then\ l\ else$ insLista(cancLista(l, p'), p, i)
- lunghezza(creaLista()) $\rightarrow$ 0
- lunghezza(insLista(l, p, i)) $\rightarrow$ lunghezza(l) + 1
- contiene(creaLista(), i') $\rightarrow$ false
- contiene(insLista(l, p, i), i') $\rightarrow$ $if\ i = i'\ then$ true $else$ contiene(l, i')

B.3 Specifica di restrizione

- leggiLista(creaLista(), p') $\rightarrow$ error
- cancLista(creaLista(), p') $\rightarrow$ error