**THE WHISPERER**

Aldo Mangione

**Introduzione**

“The Whisperer” è un’ avventura testuale in cui sei un ladro che deve recuperare un potente artefatto (il “whisperer”) dalla base di un covo di malviventi.

**Mappa**

**A screenshot of a graph

Description automatically generated with low confidence**

**Legenda:**

Lucchetto: stanza bloccata;

Punto verde: punto d’inizio;

Punto esclamativo rosso: guardia;

Chiave inglese: oggetto raccoglibile;

Stella: obiettivo;

**Come vincere**

Per vincere, si possono digitare i seguenti comandi in quest’ordine:

1. W
2. N
3. N
4. W
5. W
6. Use knife
7. S
8. S
9. W
10. W
11. N
12. N
13. Use pick
14. N
15. N
16. Pick

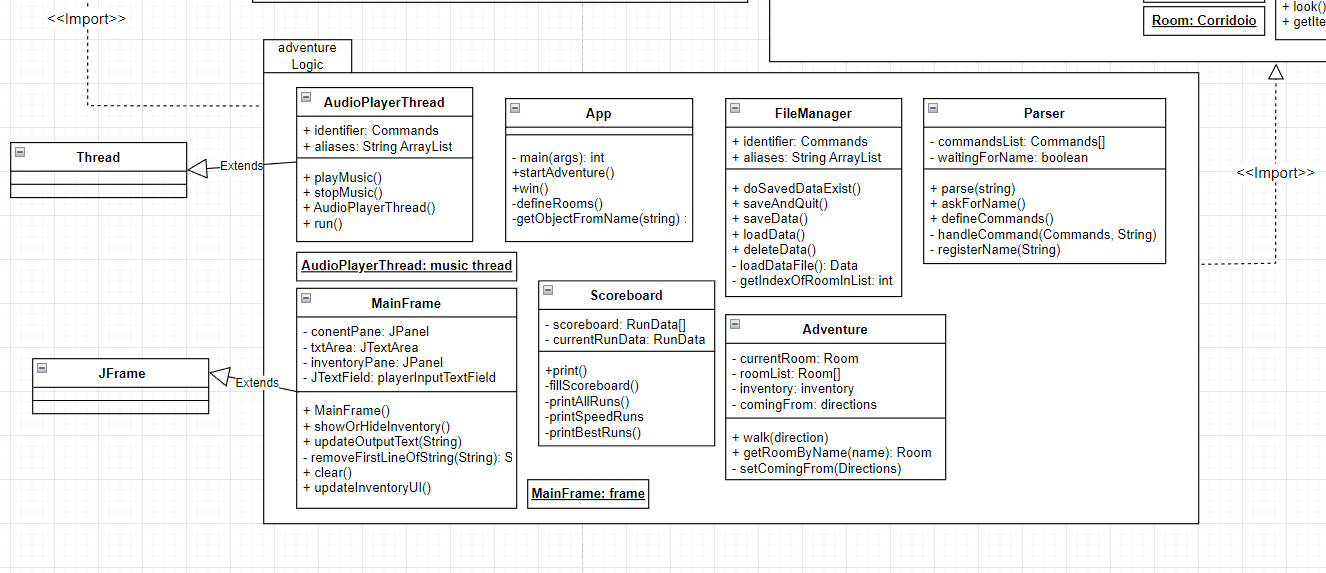
**User stories**

|  |  |  |
| --- | --- | --- |
| **Attore** | **Voglio…** | **Priorità** |
| Giocatore | Poter camminare | 2 |
| Giocatore | Poter vedere la lista dei comandi | 2 |
| Giocatore | Poter esplorare le stanze | 2 |
| Giocatore | Poter accedere ad un inventario ed usare oggetti | 1 |
| Giocatore | Poter salvare/caricare i dati | 1 |
| Giocatore | Eliminare i miei dati | 1 |
| Giocatore | Vedere il mio punteggio e quello degli altri | 1 |
| Designer | Implementare nuove stanze | 2 |
| Designer | Implementare nuovi comandi | 2 |
| Giocatore | Ascoltare della musica | 1 |
| Giocatore | Pulire la console | 1 |

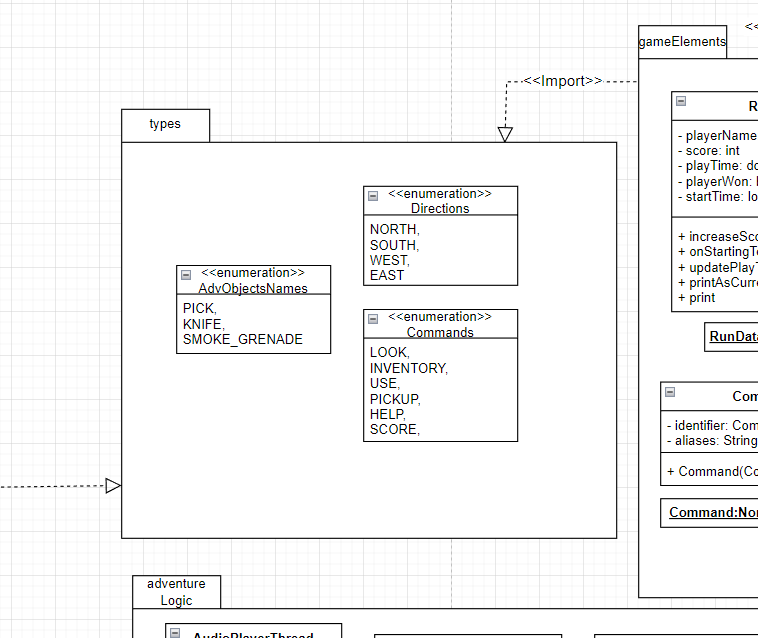
**Diagrammi delle classi**Immagine che contiene testo, interni

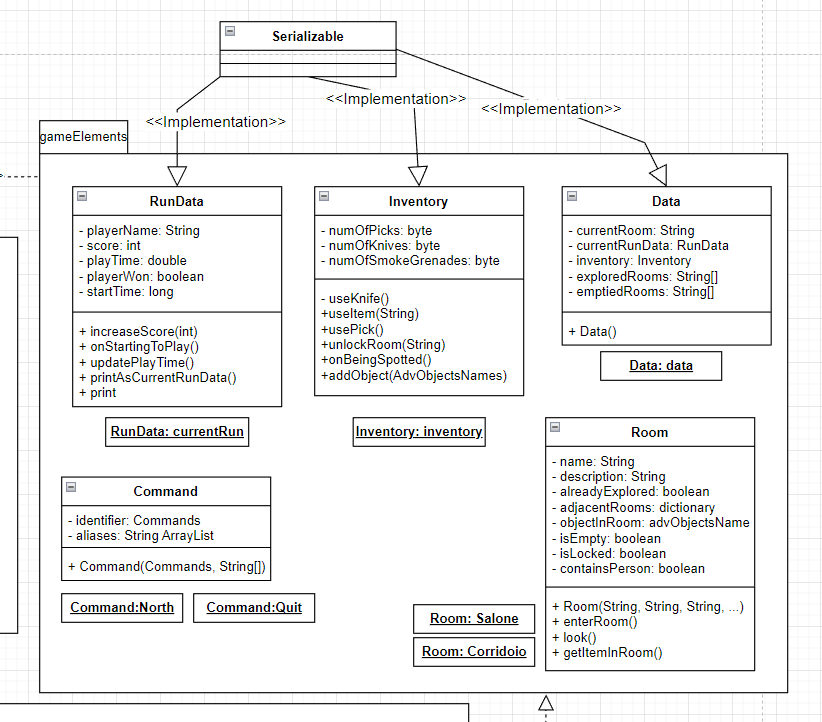
Descrizione generata automaticamente

I packages più da vicino, adventureLogic:



types (enumeratori):



gameElements: 

**Classi e funzioni**

**App.java**

Questa classe contiene il main e si occupa di avviare la partita.

**-main():** genera l’interfaccia grafica, avvia il thread della musica, e inizia la partita.

**-startAdventure():** resetta tutte le stanze, carica I dati se esistono e poi fa iniziare l’avventura;

**-win():** assegna punti, segna che hai vinto, e ti chiede il nome per salvare la run appena effettuata.

**-defineRooms():** definisce e genera le stanze;

**-getObjectFromName():** metodo usato da defineRooms() per interpretare I dati all’interno del DB.

**Parser.java**

Classe che si occupa di elaborare l’input dell’utente.

-**parse():** quando il giocatore inserisce un input e preme invio, questa funzione elabora la stringa in input preparandola per **handleCommand();**

-**askForName():** comunica al parser che la prossima stringa sarà il nome del giocatore e non un comando;

**-registerName():** imposta il nome ricevuto sulle info del giocatore;

-**handleCommand(command):** in base al comando ricevuto, richiama il codice che si occupa della richiesta

-**defineCommands():** inizializza i comandi.

**AudioPlayerThread.java**

Classe thread che si occupa di gestire la musica

-**run():** avvia il thread e riproduce la musica;

-**AudioPlayerThread**(): localizza il file della musica e lo prepara per essere riprodotto;

-**playMusic**(): se la musica è ferma, la fa riprendere;

-**stopMusic**(): se sta riproducendo la musica, la interrompe;

-**interrupt**(): interruzione del thread, fa fermare la musica ;

**FileManager.java**

Classe che si occupa di salvare e caricare i dati da file

-**doSavedDataExist():** ritorna vero se esistono file salvati;

-**saveAndQuit():** modo corretto di chiudere l’app. Salva i file prima di chiudere i file;

-**saveData**(): salva dati su file;

-**loadData():** carica i dati da file;

-**getIndexOfRoomsInRoomList**(): funzione usata da altre come metodo di trovare le stanze attraverso l’indice nella lista.

**-set…():** 4 funzioni usate per modificare la lista delle stanze in base ai dati ricevuti.

**Adventure.java**

Classe che contiene tutte le info sull’avventura in corso.

-**walk(direction):** modifica la stanza corrente procedendo nella direzione desiderata;

-**getRoomByName(name):** ritorna una stanza dalla lista delle stanze in base al nome inserito**;**

-**setComingFrom(direction):** metodo usato per stabilire la direzione in cui devi andare se non vuoi essere attaccato dalle guardie;

**MainFrame.java**

Classe che genera e gestisce l’interfaccia grafica

-**MainFrame():** crea l’interfaccia grafica.

-**showOrHideInventory():** mostra/nasconde l’inventario

-**updateOutputText( newText ):** modifica il testo mostrato aggiungendo il “newText”

-**removeFirstLineOfString( string ):** rimuove la prima riga (fino al ritorno a capo) di una stringa

-**clear():** pulisce il testo sullo schermo

**-updateInventoryUI():** aggiorna la ui dell’inventario per farla corrispondere all’inventario effettivo.

**Scoreboard.java**

Classe che gestisce il punteggio e la scoreboard

-**print():** stampa la scoreboard;

**-fillScoreboard():** metodo utilizzato per riempire la scoreboard

**-printAllRuns():** stampa tutte le info di tutte le giocate effettuate

**-printSpeedruns():** stampa le info di tutte le giocate completate in poco tempo;

**-printBestRuns():** stampa le info di tutte le giocate migliori.

-**saveRunData():** aggiunge le info sulla giocata corrente alla scoreboard.

**Command.java**

Classe che contiene le info sui comandi

-**Command(identifier, aliases):** definisce il comando

**Data.java**

Classe che viene immagazzinata su file contenente tutte le informazioni sull’ultima giocata effettuata

-**Data():** raccoglie i dati dalle altre classi

**Inventory.java**

Classe che si occupa di tenere traccia degli oggetti nell’inventario e di usarli

-**useItem(item):** elabora quale item vuole usare il giocatore ed eventualmente rimanda a **usePick()** o **useKnife();**

**-usePick():** controlla se ci sono stanze bloccate intorno al giocatore ed eventualmente le sblocca con **unlockRoom();**

-**unlockRoom(roomName):** sblocca la stanza di nome “roomName”

-**useKnife():** se ci sono persone nella stanza del giocatore, utilizza un coltello dall’inventario per rimuoverla;

-**onBeingSpotted():** se il giocatore ha fumogene, vengono usate, altrimenti questo metodo fa finire la partita;

**-addObject(obj)**: aggiunge l’oggetto all’inventario;

**Room.java**

Classe che descrive una stanza

-**Room(…):** definisce le proprietà della stanza;

-**enterRoom():** setta la stanza come stanza corrente, e stampa il testo riguardante la stanza**;**

**-getItemInRoom():** se la stanza contiene un oggetto, verrà aggiunto all’inventario del giocatore;

**RunData.java**

Classe che contiene info sulle giocate.

-**onStartingToPlay():** inizia a misurare il tempo di gioco;

-**updatePlayTime**(): aggiorna il tempo giocato;

-**printCurrentRunData**(): stampa le info nel formato “Current”;

-**print():** stampa le info nel formato giusto per la scoreboard

**Specifica struttura dati**

**Lista**

**Specifica sintattica**

**Tipi:** lista, tipoelem, intero, bool

**Operatori:**

-CREALISTA() -> lista

-LEGGILISTA(lista, intero) -> tipoelem

-SCRIVILISTA(lista, elemento,intero) -> lista

-AGGIUNGI(lista, tipoelem) -> lista

-RIMUOVI(lista, int) -> lista

-CONTIENE(lista, tipoelem) -> bool

-VUOTA(lista) -> bool

-GRANDEZZA(lista) -> int

**Specifica semantica**

**Tipi:**

-lista: collezione di n elementi dello stesso tipo ed n >= 0;

-intero: insieme dei numeri interi;

-bool: insieme dei valori {vero, falso};

**Operatori**:

-CREALISTA = l

-post: l = lista vuota <> con elementi del tipo tipoelem;

-LEGGILISTA(l, i) = elemento

-pre: l = <a\_1, …, a\_k> con i <= k;

-post: elemento = a\_i;

-SCRIVILISTA(l, e, i) = l2

-pre: l = <a\_1, … , a\_i, … , a\_k> con i <= k ed e dello stesso tipo di a;

-post: l2 = <a\_1, …, e , …, a\_k>;

-AGGIUNGI(l1, e) = l2

-pre: l1 = <a\_1, …, a\_k> con e di tipo uguale ad a appartenente ad l1 ;

-post: l2= <a\_1, …, a\_k, e>;

-VUOTA(l) = b

-pre: l = <a\_1, …, a\_k >= 0;

-post: b = vero se k > 0; b = falso altrimenti;

-CONTIENE(l, e) = b

-pre: l = <a\_1, …, a\_k> con e di tipo uguale ad a appartenente ad l ;

-post: b = vero se e è uguale ad un certo a\_i; b = falso altrimenti;

-GRANDEZZA(l) = i

-pre: l = <a\_1, …, a\_k >= 0;

-post: i = k;

-RIMUOVI(l, i) -> l2

-pre: l = <a\_1, … , a\_i, … , a\_k> con i <= k;

-post: l2 = <a\_1, …, a\_k>; (a\_i non appartiene a l2)

**Specifica algebrica**

Siano l = lista, e = elemento, i = numero naturale

CONTIENE(AGGIUNGI(l, e), e) = vero

CONTIENE(SCRIVILISTA(l, e, i), e) = vero

GRANDEZZA(CREALISTA()) = 0

LEGGILISTA(SCRIVILISTA(l, e, i), i) = e

VUOTA(CREALISTA) = vero

VUOTA(RIMUOVI(AGGIUNGI(CREALISTA, e), e) = vero

**Specifica di restrizione**

LEGGILISTA(CREALISTA(), i) = errore

SCRIVILISTA(CREALISTA(), i) = errore

RIMUOVI(CREALISTA, i) = errore