



# ASYLUM GAME

GRUPPO: Name Not Found

AUTORI

706463, Tempesta Felice.

697384, Alessandro Paparella

706467, Claudio Valenziano

## INDICE

ASYLUM GAME.....	1
INDICE.....	2
Introduzione .....	2
Idea alla base del gioco e introduzione sul progetto .....	2
Architettura del sistema e diagramma UML delle classi .....	4
Dettagli implementativi e tecnologie utilizzate .....	7
Specifica algebrica di una struttura dati utilizzata .....	13
Specifica sintattica .....	13
Specifica semantica .....	13
Soluzione di restrizione.....	13
Sezioni facoltative.....	14
Soluzione del gioco .....	14
Dettagli su compilazione/deploy del codice.....	17

## Introduzione

### Idea alla base del gioco e introduzione sul progetto

L'applicazione sarà strutturata in due diversi piani, sui quali l'utente avrà la possibilità di destreggiarsi tra enigmi(trappole, passaggi segreti...) e combattimenti con diversi nemici. Questi ultimi sono collegati da una trama che il giocatore andrà a scoprire tramite l'esperienza di gioco. Naturalmente il gioco sarà "condito" da delle difficoltà che lo renderanno tale. L'applicazione avrà un look tale da ricordare altri giochi dello stesso genere in modo da coinvolgere totalmente l'utente.

Possiamo dire che il progetto è stato ideato non solo con l'obiettivo di offrire un'avventura testuale ma anche un modulo indipendente che può essere considerato un motore di gioco ed essere utilizzato per diverse avventure testuali che si appoggiano alle risorse del modulo in questione.

Dal punto di vista sviluppativo l'idea era quella di unire vari argomenti trattati nel corso.

Argomenti trattati:

- Collection
- Generics
- Eccezioni
- Database, serializzazione e I/O
- Programmazione concorrente
- Swing
- Interfacce ed ereditarietà
- Interfacce funzionali
- Identificazione di tipo a run-time



Alcuni di questi argomenti vengono trattati con esaustività nel dettagli implementativi e tecnologie utilizzate

## Architettura del sistema e diagramma UML delle classi

I seguenti diagrammi di classe escludono dettagli poco rilevanti al fine di illustrare l'architettura del sistema come eccezioni, come metodi get e set, attributi ausiliari che si sono rivelati indispensabili ai fini dell'implementazione (come ad esempio classi per interfaccia grafica, localizzazione, attributi transient...).

### Diagramma del motore di gioco:

La classe Launcher è utilizzata come entry point per avviare le avventure testuale da parte dell'utente, ha lo scopo di istanziare la classe Engine con all'interno una classe che estende la classe astratta GameDescription: quest'ultima demanda allo sviluppatore del gioco l'implementazione dei metodi per la gestione dei comandi e dell'inizializzazione del gioco (es. lista dei comandi accettati, classe Command). La classe Engine si occupa dell'interazione con l'utente servendosi di una interfaccia Parser la quale si appoggia ad una classe ausiliaria ParserOutput per dare una rappresentazione dei comandi nel sistema.

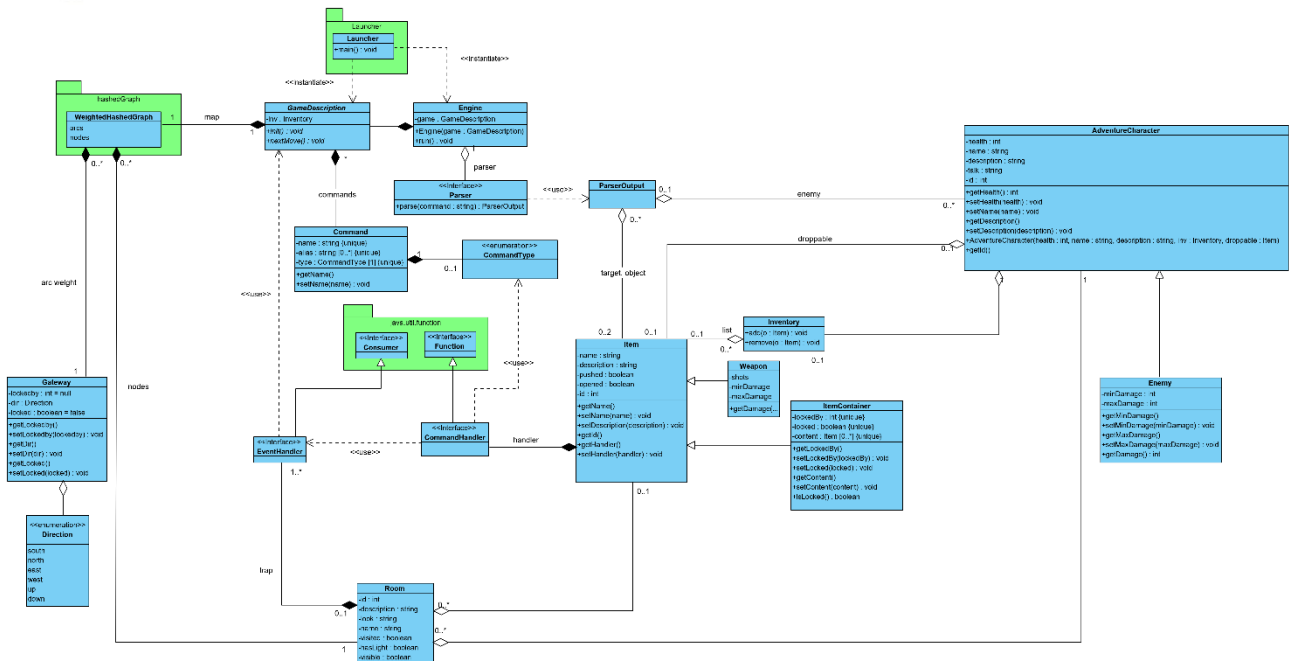
Per la rappresentazione della mappa di gioco ci siamo serviti di una struttura dati di tipo grafo presente in un package indipendente. In particolare, abbiamo utilizzato una classe che rappresenta un grafo con archi pesati e orientati, i cui nodi sono costituiti da oggetti di classe Room e gli archi all'interno della rappresentazione del gioco indicano i passaggi tra le varie stanze; questo viene reso possibile tramite la classe Gateway che contiene le direzioni dei collegamenti e indica eventuali porte bloccate.

Ogni stanza può contenere eventuali oggetti di classe Item la quale viene estesa da specializzazioni come Weapon e ItemContainer, ma anche personaggi rappresentati dalla classe.

AdventureCharacter e/o relativa specializzazione che rappresentano i nemici ostili al personaggio che, come è possibile vedere dal diagramma, possono anch'essi portare degli oggetti e un inventario analogamente al giocatore (vedi attributo inv della classe GameDescription).

Infine, ogni oggetto ha un attributo handler di tipo CommandHandler che estende un'interfaccia funzionale java Function, la quale riceve in input un parametro enumerativo CommandType in base al quale restituisce un EventHandler che estende l'interfaccia Consumer che modificherà lo stato del gioco; entrambi le classi EventHandler e CommandHandler sono classi anonime che vengono dichiarate in cui ogni dichiarazione di istanza può adottare comportamenti diversi a seconda dei casi (sovrascrivendo i metodi apply() e accept()).

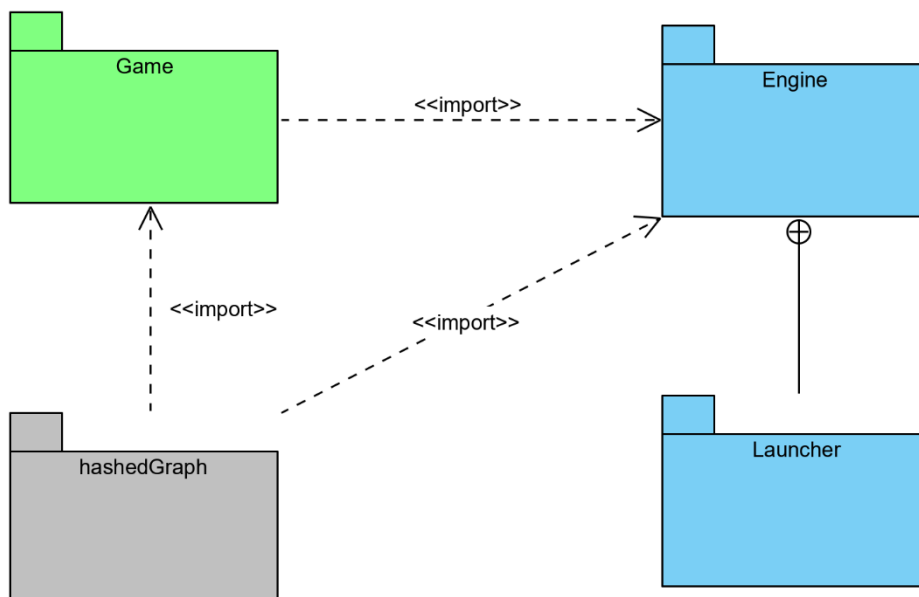
Inoltre, ogni stanza contiene un attributo trap usato per gestire l'eventuale presenze di trappole, anch'esse gestite attraverso l'interfaccia funzionale EventHandler.



### Diagramma del package di progetto:

Il seguente diagramma di packages illustra l'architettura del sistema, in particolare possiamo notare come il package Engine ha al suo interno il Launcher dedicato al caricamento dei giochi tramite files Jar con l'ausilio di un'interfaccia grafica. Il package Game ha la funzione di contenere le classi create ad hoc per l'implementazione di una vera e propria avventura testuale, in particolare deve obbligatoriamente contenere una classe che estenda GameDescription presente in Engine.

Entrambi i packages Engine ed Game si servono del package hashedGraph per la realizzazione della mappa di gioco.



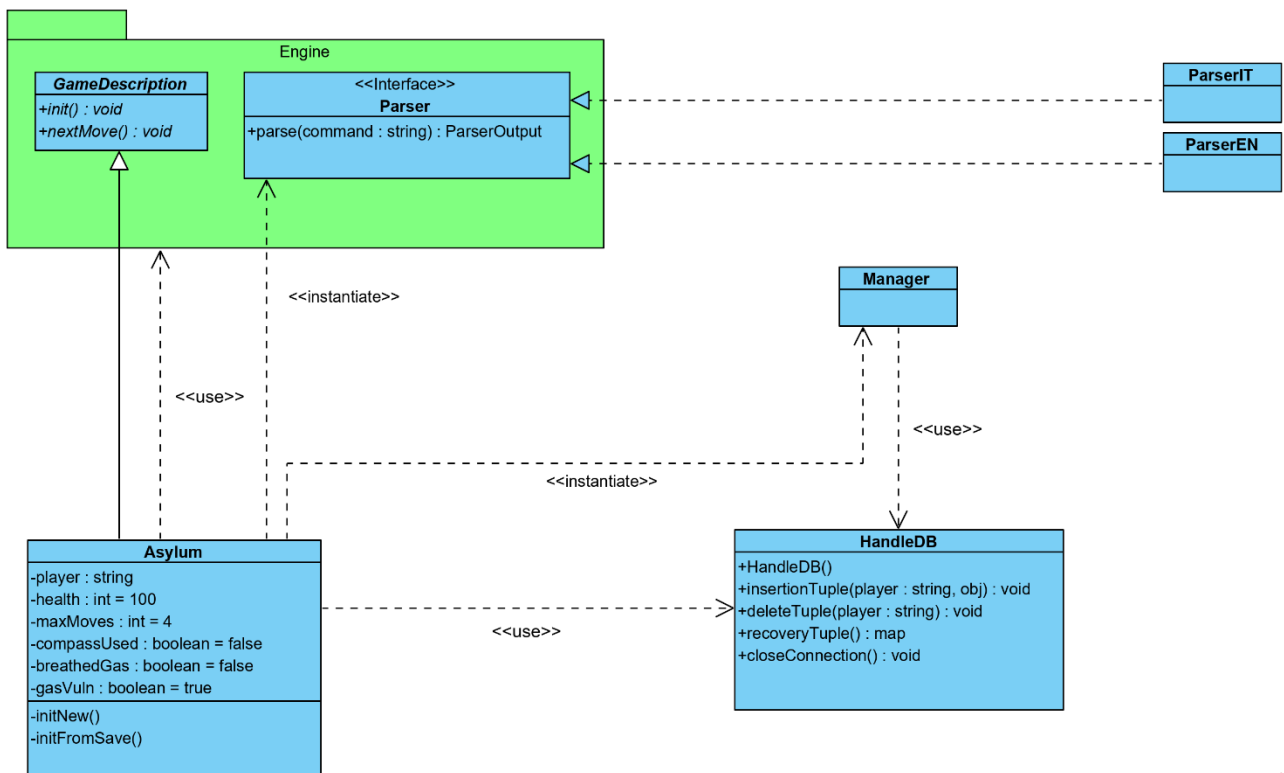


### Diagramma del gioco implementato:

Il seguente diagramma rappresenta il caso specifico del nostro progetto, il quale è un gioco multilingua per questo motivo sono state realizzate due implementazioni di parser in italiano e in inglese.

La scelta della lingua è gestita attraverso la classe manager che consiste in una interfaccia grafica con la quale l'utente crea o gestisce i salvataggi precedenti, questo è reso possibile dalla classe HandleDB che si occupa di fornire una gestione semplificata dal database con metodi strettamente necessari ai nostri scopi.

La classe Asylum è la classe usata per implementare la nostra avventura testuale che estende GameDescription, inoltre, contiene attributi specifici delle meccaniche dell'avventura realizzata, in questa classe vengono dichiarate tutte le classi anonime riguardanti oggetti e trappole.





## Dettagli implementativi e tecnologie utilizzate

### Generics e collection

Sono state utilizzate per la implementazione del grafo pesato e nei relativi metodi:

```
public class WeightedHashedGraph<T, W> implements Serializable {  
    private static final long serialVersionUID = 7371846499712012234L;  
    private Map<T, Map<T, W>> arcs = new HashMap<T, Map<T, W>>();  
    private Set<T> nodes= new HashSet<T>();  
  
    public void insArc(T start, T end, W weight) {  
        if(!this.contains(start)) {  
            this.insNode(start);  
        }  
        if(!this.contains(end)) {  
            this.insNode(end);  
        }  
        arcs.get(start).put(end, weight);  
    }  
}
```

Il placeholder T corrisponde alle istanze di Room, mentre, il placeholder W corrisponde alle istanze dei gateway.

Ci siamo serviti delle implementazioni di Map e Set che si servono del meccanismo di Hashing.

Nel esempio seguente è mostrata l'implementazione del metodo di hashing per gli oggetti di tipo Room utilizzando l'attributo numerico id

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + ((id == null) ? 0 : id.hashCode());  
    return result;  
}
```

### Database serializzazione e I/O

La gestione del db è stata realizzata attraverso la classe HandleDB usando l'interfaccia DriverManager e servendosi del database del tipo stand alone. Il database viene utilizzato per la gestione dei salvataggi dello stato del gioco. Di seguito è mostrato il metodo costruttore per la creazione della base di dati, in seguito un metodo per l'inserzione di un salvataggio sullo stato del gioco, costituito da: nome giocatore, data corrente, e un oggetto "obj" che viene serializzato e contiene il riferimento alla istanza di gioco.

```
public HandleDB() throws SQLException, Exception {  
    Class.forName("org.h2.Driver");  
    conn = DriverManager.getConnection("jdbc:h2:~/saves");  
    final String CREATE_TABLE = "CREATE TABLE IF NOT EXISTS saves (player  
        VARCHAR(30) PRIMARY KEY,"  
        + " day VARCHAR(11),"  
        + " game OBJECT )";  
    Statement stm = conn.createStatement();  
    stm.executeUpdate(CREATE_TABLE);  
    stm.close();  
}
```

```
public void insertionTuple(String player, Object obj) throws SQLException {  
    PreparedStatement pstmt = conn.prepareStatement(  
        "INSERT INTO saves "
```



```
        + "VALUES (?, ?, ?)");  
    pstmt.setString(1, player);  
    pstmt.setString(2, LocalDate.now().toString());  
    pstmt.setObject(3, obj);  
    pstmt.executeUpdate();  
    pstmt.close();  
}
```

Librerie utilizzate per la serializzazione e il corretto utilizzo della base di dati

```
import java.io.  
import java.sql
```

A tal scopo ogni classe presente nella struttura del gioco implementa l'interfaccia Serializable:

```
public class Item implements Serializable {  
    private static final long serialVersionUID = 5059157097169486201L;  
public class Asylum extends GameDescription implements Serializable {  
    private static final long serialVersionUID = -78135229798072209L;
```

All'avvio dell'applicazione si richiede all'utente di avviare una nuova partita o di caricare una recente, nel caso della nuova partita il sistema andrà ad inserire nel db una tupla contenente il nome del giocatore e lo stato del gioco.

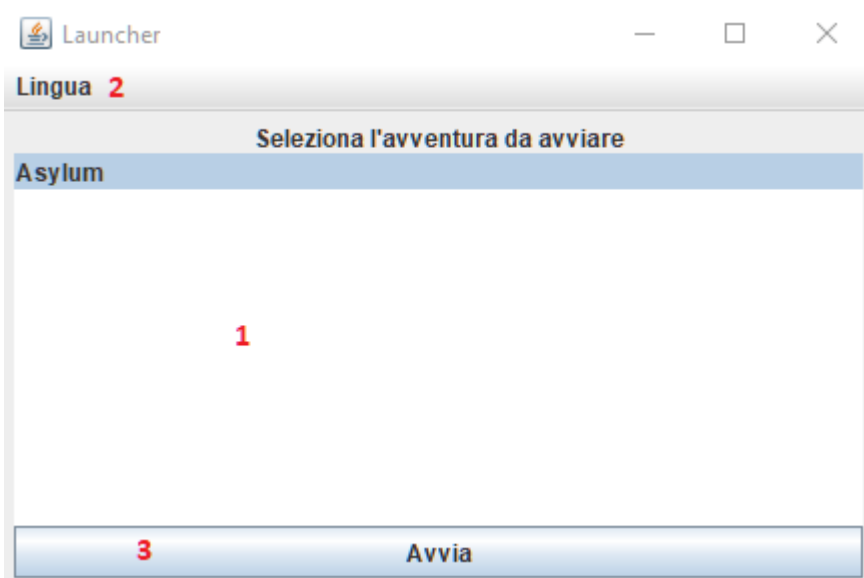
Mentre come vediamo nell'else nel caso di caricamento il giocatore andrà a scegliere il salvataggio da caricare tramite il metodo `initFromSave(Asylum)` il quale carica lo stato del gioco del caricamento nell'istanza attuale.

```
    if(frame.getSave()==null) {  
        initNew();  
        this.db.insertionTuple(this.player, this);  
    }else {  
        initFromSave((Asylum) frame.getSave());  
        if(!this.lang.equals(Manager.locale)) {  
            this.lang=Manager.locale;  
            loadLocales();  
        }  
    }  
}  
private void initFromSave(Asylum save) throws SQLException, Exception {  
    this.breathedGas = save.breathedGas;  
    this.gasVuln = save.gasVuln;  
    this.health = save.health;  
    this.maxMoves = save.maxMoves;  
    this.player = save.player;  
    this.compassUsed = save.compassUsed;  
    this.setMap(save.getMap());  
    this.setInventory(save.getInventory());  
    this.setCurrentRoom(save.getCurrentRoom());  
    this.setCurrentEnemy(save.getCurrentEnemy());  
    this.setCommandTarget(save.getCommandTarget());  
    this.setCommands(save.getCommands());  
    this.lang = save.lang;  
}
```



## Interfacce swing, thread

Per la realizzazione delle interfacce grafiche ci siamo serviti della libreria `javax.swing`



1. In questa sezione è presente la corrispondenza all'oggetto `JList`
2. In questa sezione è presente la corrispondenza all'oggetto `JMenuBar`
3. In questa sezione è presente la corrispondenza all'oggetto `JButton`

Qui di seguito è riportato l'utilizzo di un thread per la gestione della finestra, la programmazione concorrente viene utilizzata nella gestione delle interfacce grafiche tramite l'utilizzo dell'interfaccia `Runnable`.

```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                Launcher frame = new Launcher();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

Librerie utilizzate per questa gestione `java.awt`

## Interfacce funzionali

Per quanto riguarda le interfacce funzionali sono state realizzate le interfacce EventHandler e CommandHandler che estendono rispettivamente le interfacce Consumer e Function. Si è rivelato indispensabile la definizione di queste due nuove interfacce dal momento che ai fini del salvataggio del gioco era necessario avere delle interfacce funzionali che implementassero anche l'interfaccia Serializable

Ai fini di riutilizzo del codice abbiamo ritenuto di ridefinire i metodi standar pickUp() e drop()

```
public interface EventHandler extends Consumer<GameDescription>, Serializable {  
    public static void pickUp(Item i, GameDescription g) throws InvalidCommandException {  
        if(g.getCurrentRoom().getObjects().contains(i)) {  
            g.getInventory().add(i);  
            g.getCurrentRoom().getObjects().remove(i);  
        } else throw new InvalidCommandException();  
        for(Item it : g.getCurrentRoom().getObjects()) {  
            if(it instanceof ItemContainer && ((ItemContainer)it).getContent().contains(i)) {  
                ((ItemContainer)it).remove(i);  
            }  
        }  
    }  
    public static void drop(Item i, GameDescription g) {  
        g.getInventory().remove(i);  
        g.getCurrentRoom().getObjects().add(i);  
    }  
}
```

```
public interface CommandHandler extends Function<CommandType, EventHandler>, Serializable {  
}
```

Di seguito è riportato un esempio di dichiarazione di classe anonima EventHandler che costituisce una trappola posta in una stanza del gioco, la quale va a modificare come si nota la struttura della mappa bloccando una porta facendo rimanere in trappola il giocatore

```
paddedCell.setTrap(new EventHandler() {  
    @Override  
    public void accept(GameDescription t) {  
        // TODO Auto-generated method stub  
        if (t.getCurrentRoom().hasLight() && t.getCurrentEnemy()!=null) {  
            try {  
                t.getMap().readArc(paddedCell, surveillance).setLockedBy(key_1.getId());  
                t.getMap().readArc(paddedCell, surveillance).setLocked(true);  
                paddedCell.setDescription(ResourceBundle.getBundle("trap", ((Asylum)t).lang).getString("descr_room_17_light"));  
                paddedCell.setLook(ResourceBundle.getBundle("trap", ((Asylum)t).lang).getString("look_room_17_light"));  
                System.out.println(ResourceBundle.getBundle("trap", ((Asylum)t).lang).getString("trapped"));  
            } catch (Exception e) {  
                System.out.println(e.getMessage());  
            }  
        }  
        if (t.getCurrentRoom().hasLight() && t.getCurrentEnemy()==null) {  
            paddedCell.setDescription(ResourceBundle.getBundle("trap", ((Asylum)t).lang).getString("descr_room_17_no_enemy"));  
        }  
    }  
});
```

Di seguito è riportato un esempio di dichiarazione di classe anonima CommandHandler che attraverso il suo caso di uso come si può notare va a disattivare la trappola mortale per il personaggio.

```
final Item keypad = new Item(i.getString("name_item_16"), i.getString("descr_item_16"), null);
keypad.setHandler(new CommandHandler() {

    @Override
    public EventHandler apply(CommandType t) {
        switch(t) {
            case LOOK_AT:
                return new EventHandler() {
                    @Override
                    public void accept(GameDescription t) {
                        // TODO Auto-generated method stub
                        System.out.println(keypad.getDescription());
                    }
                };
            case USE:
                return new EventHandler() {
                    @Override
                    public void accept(GameDescription t) {
                        // TODO Auto-generated method stub
                        if (hallway3.getTrap()!=null){
                            if(!keypad.isPushed()) {
                                Scanner scan = new Scanner(System.in);
                                System.out.println(":");
                                String codEntered = scan.nextLine();
                                String[] tokens = codePaper.getDescription().split("\\s+");
                                if(codEntered.equals(tokens[3])) {
                                    hallway3.setTrap(null);
                                    System.out.println(ResourceBundle.getBundle("item", ((Asylum)t).lang).getString("accepted_item_16"));
                                    keypad.setPushed(true);
                                } else System.out.println(ResourceBundle.getBundle("item", ((Asylum)t).lang).getString("error_item_16"));
                            } else System.out.println(ResourceBundle.getBundle("item", ((Asylum)t).lang).getString("trap_off"));
                        } else System.out.println(ResourceBundle.getBundle("item", ((Asylum)t).lang).getString("trap_off"));
                    }
                };
            default:
                return invalidCommand;
        }
    }
});
```

### Identificazione di tipo a run-time

Nel sistema sono presenti tecniche di identificazione a run time di meta classi con RTTI. Tradizionale nel metodo pickUp() illustrato precedentemente, il quale controlla se ci sono istanza di ItemContainer nella stanza per rimuovere l'oggetto ricercato. È presente il meccanismo di riflessione per caricare da un pacchetto jar esterno la classe riguardante una specifica avventura testuale ricavandone in maniera dinamica il costruttore.



```
private void launchButtonActionPerformed() throws LoaderException, Exception {
    String selected = list.getSelectedValue();
    URLClassLoader loader = URLClassLoader.newInstance(new URL[] {games.get(selected).toURI().toURL()});
    try {
        Class c = loader.loadClass("game."+selected);
        if(!c.getGenericSuperclass().equals(GameDescription.class)) {
            throw new LoaderException(bundle.getString("nogamefound"));
        }
        final GameDescription g = (GameDescription) c.getConstructor().newInstance();
        Thread t = new Thread( new Runnable() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                Engine engine = new Engine(g);
                try {
                    engine.run();
                } catch (Exception e) {
                    // TODO: handle exception
                }
            }
        });
        t.start();
    } catch (ClassNotFoundException e) {
        throw new LoaderException(bundle.getString("nogamefound"));
    }
}
```

### Tecnologie utilizzate

Per ottenere un sistema multilingua abbiamo deciso di avvalerci delle classi Locale e ResourceBundle, per ottenere gli output localizzati in maniera dinamica

```
ResourceBundle r = ResourceBundle.getBundle("room", lang);
Room room1 = new Room(r.getString("descr_room_1"),r.getString("look_room_1"),r.getString("name_room_1"));
Room room2 = new Room(r.getString("descr_room_2"),r.getString("look_room_2"),r.getString("name_room_2"));
```

Dove si può notare che le risorse vengono prese da un file di tipo .properties facendo riferimento al prefisso, in questo caso room: ad esempio se selezionata la lingua italiana room\_it.properties. Successivamente si utilizza questo riferimento al file per ottenere risorse localizzate attraverso un meccanismo di chiave-valore.

## Specifica algebrica di una struttura dati utilizzata

### Specifica sintattica

Sorts: WeightHashedGraph, nodes, boolean, set, weight

- NewWeightHashedGraph() -> WeightHashedGraph
- isEmpty(WeightHashedGraph) -> Boolean
- insNode(node, WeightHashedGraph) -> WeightHashedGraph
- removeNode(node, WeightHashedGraph) -> WeightHashedGraph
- getAdjacents(node, WeightHashedGraph) -> set
- contains(node, WeightHashedGraph) -> Boolean
- containsArc(node, node, WeightHashedGraph) -> boolean
- insArc(node, node, weight, WeightHashedGraph) -> WeightHashedGraph
- removeArc(node, node, WeightHashedGraph) -> WeightHashedGraph
- readArc(node, node, WeightHashedGraph) -> WeightHashedGraph

### Specifica semantica

Declare g : WeightHashedGraph , n1,n2 : Node, w,w2 : weight

- NewWeightHashedGraph() = g
- isEmpty(NewWeightHashedGraph) = true
- isEmpty(insArc(n1, n2, w, g)) = false
- readArc(n1, n2, insArc(n1, n2, w2, insArc(n1, n2, w, g))) = w2
- readArc(n1, n2, insArc(n1, n2, w, g)) = w
- removeNode(n1, insNode(n1, NewWeightHashedGraph)) = NewWeightHashedGraph
- contains(n1, removeArc(n1, n2, insArc(n1, n2, w, NewWeightHashedGraph))) = true
- contains(n1, insNode(n1, NewWeightHasheGraph)) = true
- contains(n1, NewWeightHasheGraph) = false
- contains(n1,insArc(n1, n2, w, NewWeightHashedGraph)) = true
- containsArc(n1, n2, insArc(n1, n2, w, g)) = true
- containsArc(n2, n1, insArc(n1, n2, w, g)) = false
- containsArc(n1, n2, removeNode(n1, insArc(n1, n2, w, NewWeightHashedGraph)) = false
- getAdjacents(n1, insArc(n1, n2, w, g)) = n2
- getAdjacents(n1, insArc(n2, n1, w, g)) =  $\emptyset$
- getAdjacents(n1, insArc(n1, n1, w, g)) = n1

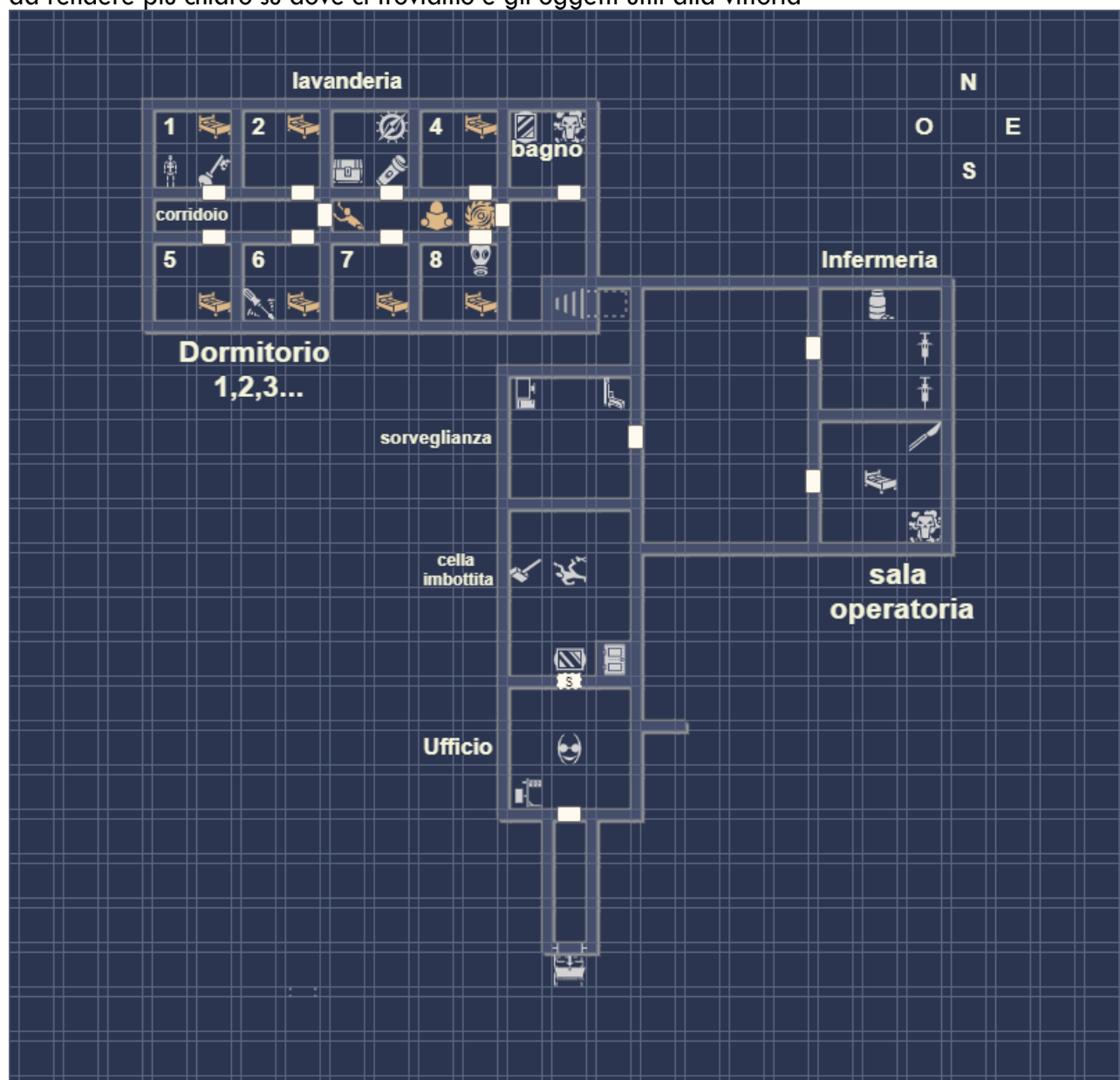
### Soluzione di restrizione

- readArc(n1, n2, NewWeightHashedGraph) = error
- getAdjacents(n1, n2, NewWeightHashedGraph) = error
- containsArc(n1, n2, NewWeightHashedGraph) = error



## Sezioni facoltative

### *Soluzione del gioco*

Come possiamo vedere qui è riportata la mappa con i relativi oggetti presenti nelle stanze, in modo da rendere più chiaro su dove ci troviamo e gli oggetti utili alla vittoria



Di seguito si riportano i comandi e i loro alias utilizzabili durante l'esperienza di gioco

Alias per utilizzarli		
Lingua	Italiano 	Inglese 
<b>DROP</b>	lascia getta scarta	drop discard throw
<b>TURN ON</b>	accendi	turn_on
<b>TURN OFF</b>	spegni	turn_off
<b>USE</b>	usa utilizza	use utilize
<b>WALK TO</b>	vai entra	go enter
<b>TALK TO</b>	parla dialoga interagisci comunica	talk converse interact communicate
<b>BREAK</b>	rompi distruggi spacca colpisci attacca	break destroy split attack
<b>CLOSE</b>	chiudi	close
<b>OPEN</b>	apri	open
<b>PICK UP</b>	raccogli prendi	pick take
<b>LOOK AT</b>	osserva guarda vedi trova cerca descrivi	look see observe watch find describe
<b>END</b>	end fine esci muori ammazzati ucciditi suicidati exit	end exit die suicide quit
<b>INVENTORY</b>	Inventario inv i l	inventory inv l i
<b>UP</b>	sopra sali	up above climb
<b>DOWN</b>	giu sotto scendi	down below come_downest
<b>EAST</b>	est e E Est EST	east e E East EAST
<b>WEST</b>	ovest o O Ovest OVEST	west w W West WEST
<b>NORTH</b>	nord n N Nord NORD	north n N North NORTH
<b>SOUTH</b>	sud s S Sud SUD	south s S South SOUTH
<b>SOUTH EAST</b>	sud_est se SE Se sE Sud_est Sud_Est south_Est SUD_EST	south_east se SE Se sE South_east south_East south_East SOTUH_EAST
<b>SOUTH WEST</b>	sud_ovest so SO So sO Sud_west Sud_Ovest south_Ovest SUD_OVEST	south_west sw SW Sw sW South_west south_West south_West SOUTH_WEST
<b>NORTH EAST</b>	nord_est ne NE Ne nE Nord_est Nord_Est nord_Est NORD_EST	north_east ne NE Ne nE North_east north_East north_East NORTH_EAST
<b>NORTH WEST</b>	nord_ovest no NO No nO Nord_west Nord_Ovest nord_Ovest NORD_OVEST	north_west nw NW Nw nW North_west north_West north_West NORTH_WEST

### Suggerimenti :

Per vincere non bisogna entrare in tutte le stanze, ad esempio nelle stanze dormitorio 2,5,4,7 non sono presenti oggetti utili durante il gioco.

Bisogna osservare bene tutti gli oggetti che si trovano lungo il percorso in modo da capire la trama e di proseguire durante il gioco con facilità.

Nel dormitorio 8 è presente la maschera anti-gas la quale protegge dal gas e quindi rende anche inutile l'utilizzo delle pillole per curarci, dopo essere stati colpiti dal gas avremo 4 comandi validi da utilizzare prima di morire.

Appena si arriva nel corridoio 4 verrà effettuato un salvataggio automatico il quale andrà a conservare lo stato del gioco.

Se si vorrà uscire dal gioco basta dare il comando

➤ End o i suoi relativi alias

Qui vengono riportati i comandi da eseguire nelle stanze e vincere

### Dormitorio 1:

- Osserva cadavere
- Prendi chiave
- Usa chiave

- Vai corridoio 1

**Corridoio 1:**

- Vai dormitorio 6

**Dormitorio 6:**

- Prendi cacciavite (utile per eliminare il nemico che troveremo)

- Vai corridoio 1

**Corridoio 1:**

- Vai corridoio 2

**Corridoio 2:**

- Usa cacciavite (fino ad ucciderlo)
- Prendi foglio
- Vai lavanderia

**Lavanderia:**

- Apri cassa
- Osserva cassa
- Prendi bussola
- Prendi torcia
- Vai corridoio 2

**Corridoio 2:**

- Osserva foglio
- Usa tastierino ed inserire 5030
- Vai corridoio 3

**Corridoio 3:**

- Vai corridoio 4

**Corridoio 4:**

- Vai infermeria

**infermeria:**

- Prendi pillole
- Prendi adrenalina x2
- Usa adrenalina x2
- Vai corridoio 4

**Corridoio 4:**

- Vai Sala operatoria

**Sala Operatoria:**

- Prendi bisturi
- Vai corridoio 4

**Corridoio 4:**

- Usa pillole
- Vai sorveglianza

**Sorveglianza:**

- Prendi pistola
- Vai cella imbottita

**Cella imbottita:**

- Accendi torcia
- Usa pistola (fino ad ucciderlo)
- Rompi specchio
- Vai ufficio





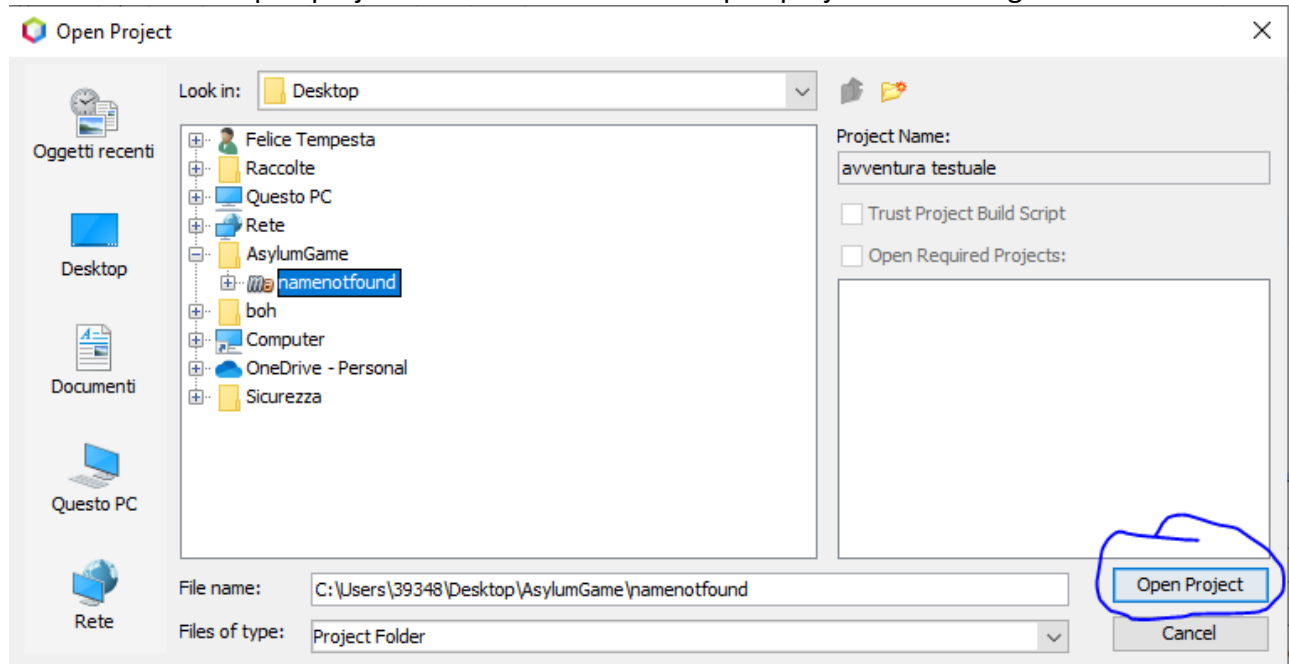
**Ufficio:**

- Usa pistola/bisturi (fino ad ucciderlo)
- Vai uscita

*Dettagli su compilazione/deploy del codice*

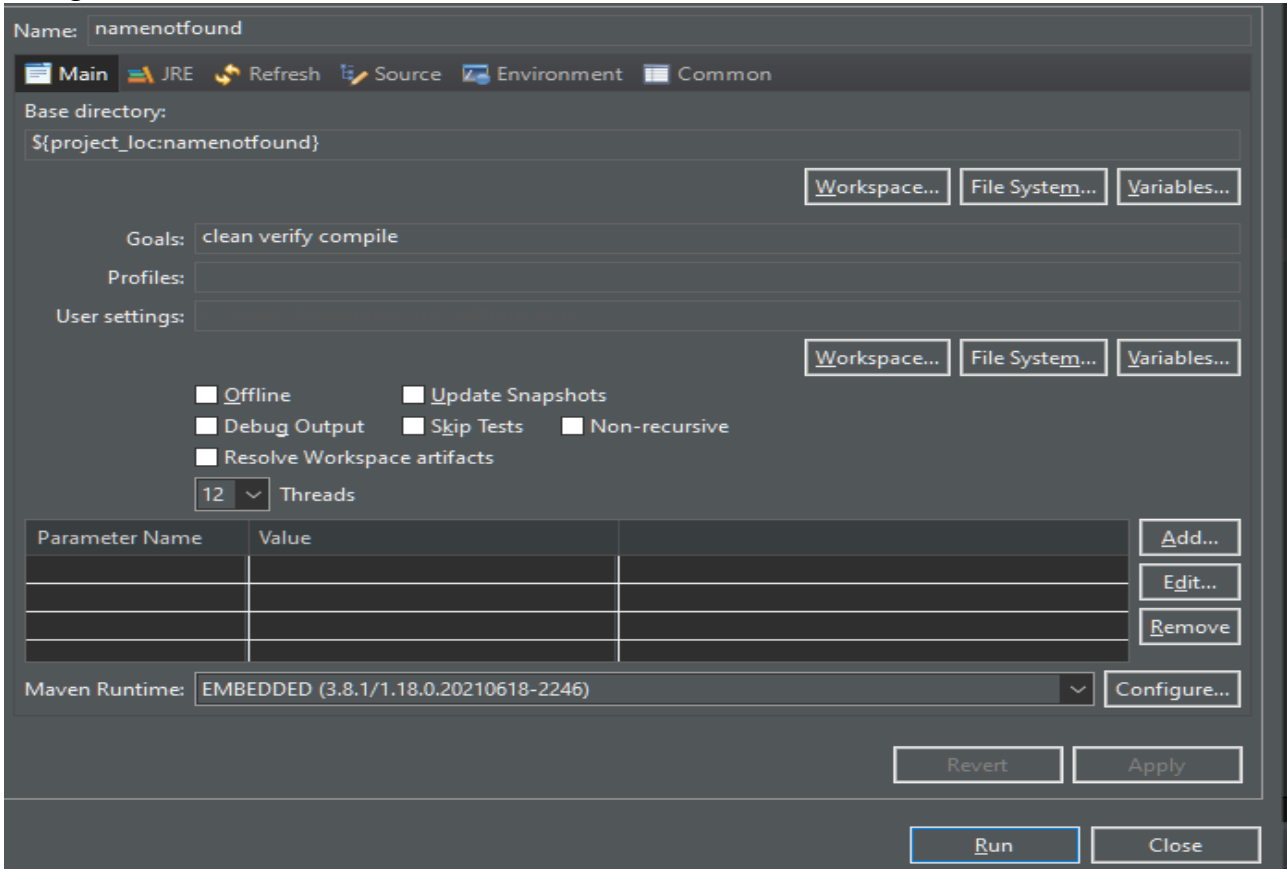
Per una corretta compilazione del codice su NetBeans 12.5

Andare su File -> Open project -> selezionare il file -> Open project come in figura



Per ide Eclipse:

Tasto destro sulla cartella del progetto -> run configuration -> maven build -> new run configuration e impostare i seguenti parametri, successivamente eseguire il run con la configurazione creata



Name: namenotfound

Main JRE Refresh Source Environment Common

Base directory: \${project\_loc:namenotfound}

Workspace... File System... Variables...

Goals: clean verify compile

Profiles:

User settings:

Workspace... File System... Variables...

☐ Offline ☐ Update Snapshots

☐ Debug Output ☐ Skip Tests ☐ Non-recursive

☐ Resolve Workspace artifacts

12 Threads

Parameter Name	Value

Add... Edit... Remove

Maven Runtime: EMBEDDED (3.8.1/1.18.0.20210618-2246) Configure...

Revert Apply

Run Close