# La mia ragazza è pazza!

**Componenti del gruppo:**

Storelli Leonardo

Rella Nicolò

Nacci Mario

# Indice

1.Breve descrizione del caso di studio:..... 3
2.Diagramma delle classi:..... 4
3.Specifica algebrica della classe Inventory: ..... 7
4.Applicazione argomenti: ..... 8
4.1 Collection: ..... 8
4.2 Eccezioni: ..... 8
4.3 File ..... 8
4.4 Database: ..... 9
4.5 Threads:..... 15
4.6 Socket: ..... 17
4.7 lambda expressions:..... 22
4.8 Swing:..... 23
5.Guida per finire il gioco:..... 27

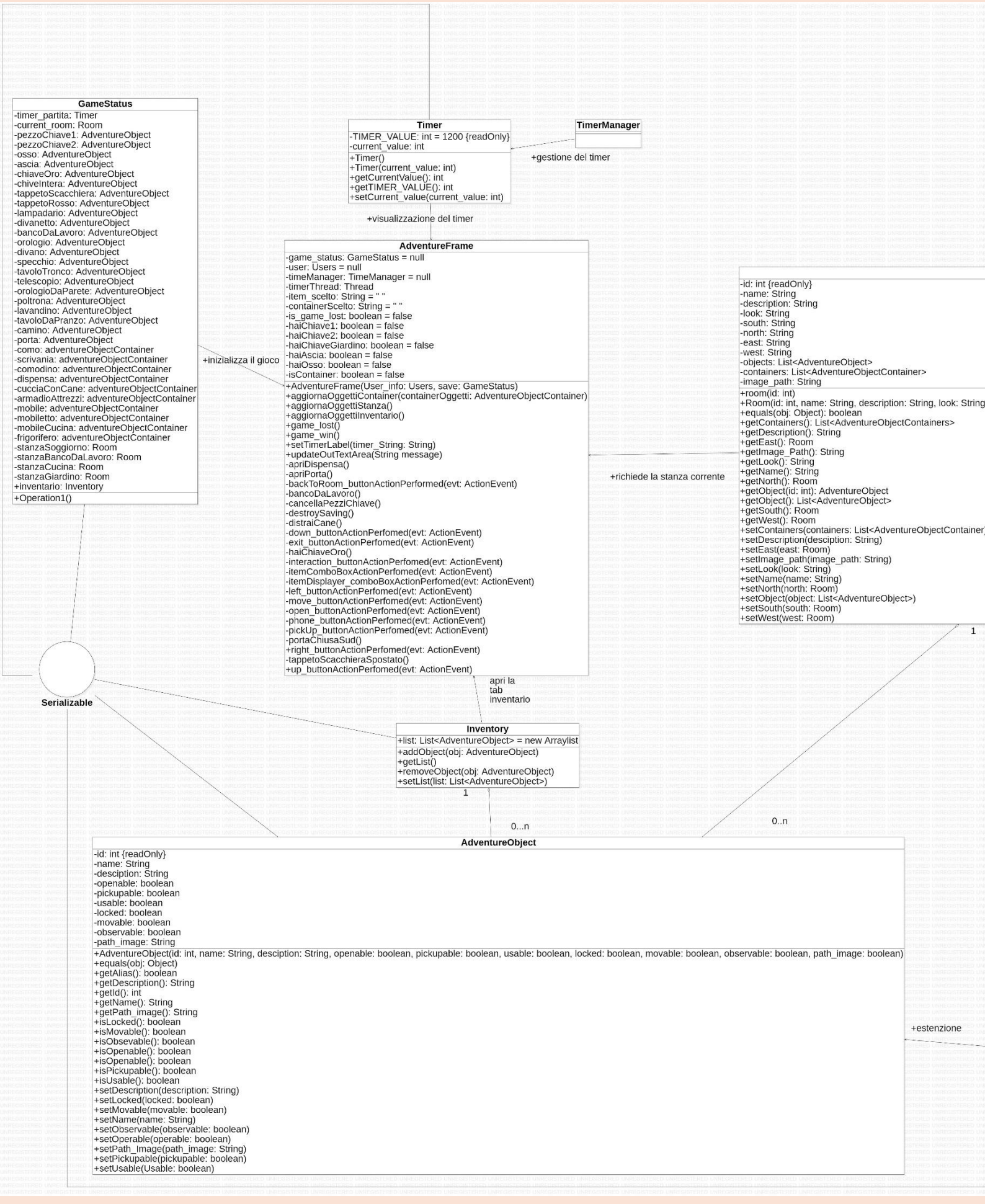
## 1.Breve descrizione del caso di studio:

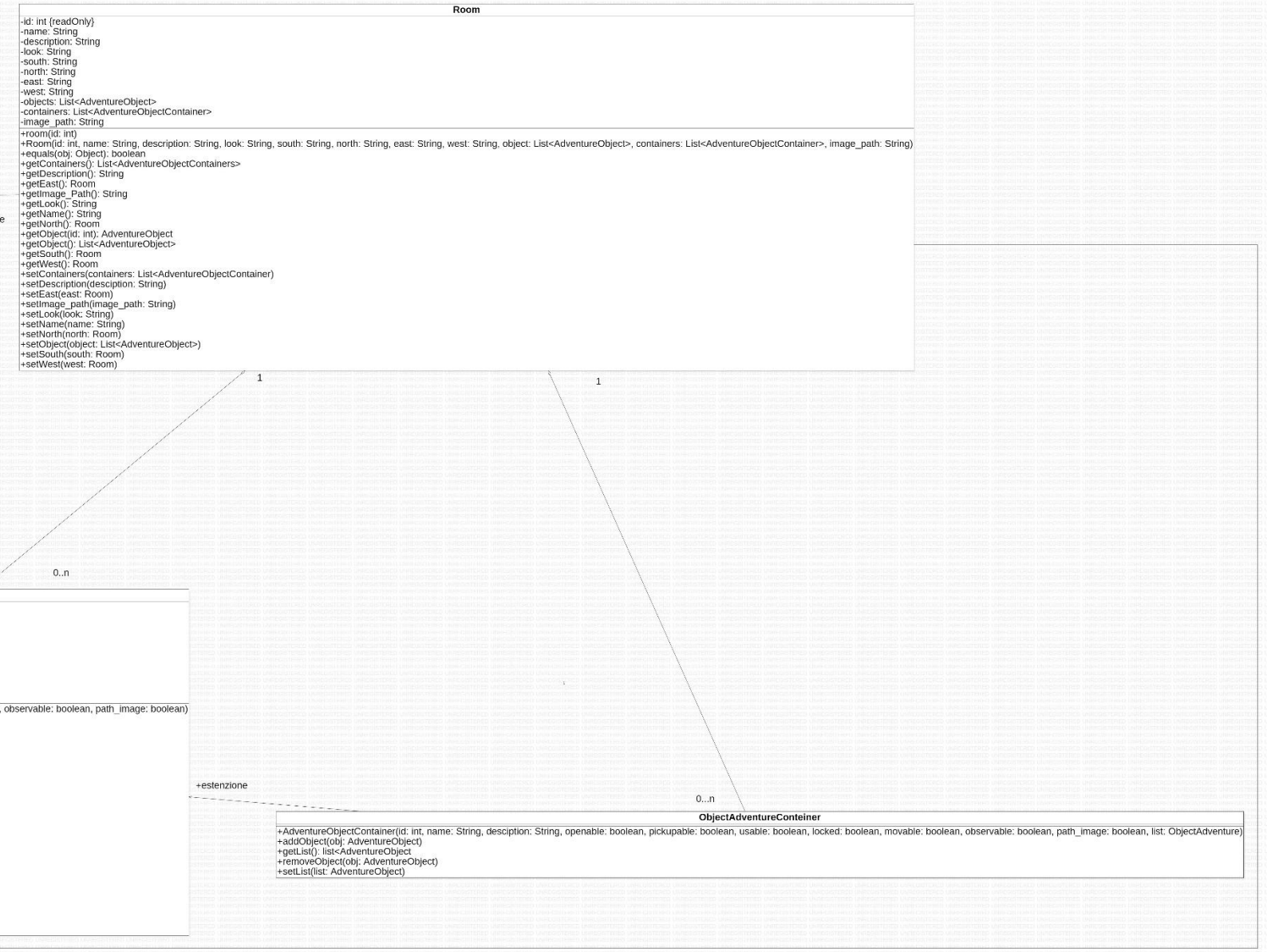
Questo caso di studio è un videogioco fatto in Java utilizzando le nozioni apprese durante il corso. Il videogioco in questo è un'avventura grafica chiamata: "La mia ragazza è pazza!" dove impersoneremo il protagonista che dovrà scappare dalla casa della propria consorte in tempo prima che ritorni per ammazzarlo.

Per fuggire di casa il protagonista dovrà esplorare le varie stanze alla ricerca di oggetti utili per superare delle prove che alla fine li permetterà di trovare la chiave d'oro per riuscire nell'impresa.

# 2.Diagramma delle classi:

qui è riportato il diagramma UML della classe *AdventureFrame*




Room
-id: int {readOnly}
-name: String
-description: String
-look: String
-south: String
-north: String
-east: String
-west: String
-objects: List<AdventureObject>
-containers: List<AdventureObjectContainer>
-image_path: String
+room(id: int)
+Room(id: int, name: String, description: String, look: String, south: String, north: String, east: String, west: String, object: List<AdventureObject>, containers: List<AdventureObjectContainer>, image_path: String)
+equals(obj: Object): boolean
+getContainers(): List<AdventureObjectContainers>
+getDescription(): String
+getEast(): Room
+getImage_Path(): String
+getLook(): String
+getName(): String
+getNorth(): Room
+getObject(id: int): AdventureObject
+getObject(): List<AdventureObject>
+getSouth(): Room
+getWest(): Room
+setContainers(containers: List<AdventureObjectContainer)
+setDescription(desciption: String)
+setEast(east: Room)
+setImage_path(image_path: String)
+setLook(look: String)
+setName(name: String)
+setNorth(north: Room)
+setObject(object: List<AdventureObject>)
+setSouth(south: Room)
+setWest(west: Room)
1
1
0..n
, observable: boolean, path_image: boolean)
+estenzione
0...n
ObjectAdventureConteiner
+AdventureObjectContainer(id: int, name: String, desciption: String, openable: boolean, pickupable: boolean, usable: boolean, locked: boolean, movable: boolean, observable: boolean, path_image: boolean, list: ObjectAdventure)
+addObject(obj: AdventureObject)
+getList(): list<AdventureObject
+removeObject(obj: AdventureObject)
+setList(list: AdventureObject)

Questo diagramma delle classi rappresenta il funzionamento della classe AdventureFrame. Dopo aver effettuato il login, la classe GameStatus inizializza gli oggetti di tipo AdventureObject, AdventureObjectConteiner e Room, in caso in cui non ci siano salvataggi salvati nel db, altrimenti prenderà quel salvataggio e lo caricherà.  AdventureFrame permette anche di visualizzare il Timer che è gestito dalla classe TimeManager. Questa classe gestisce anche gli spostamenti nelle stanze tramite i bottoni presenti nell'interfaccia per poi tramite gli attributi dell'oggetto di classe Room che è attualmente visualizzato sovrascrive i valori della stanza corrente con la nuova stanza. Nella classe Adventureframe sono anche presenti metodi che controllano le condizioni di vittoria(game_win) e sconfitta del giocatore(game_lost).

# 3.Specifica algebrica della classe Inventory:

**Specifica Sintattica:**

**Sort:** list, object, inventario

**Operazioni:**

| | |
|---|---|
| Inventory () ->inventario | crea un nuovo inventario |
| getList(inventario)->list | prende un inventario |
| addObject(inventario, Object) -> inventario | aggiunge un elemento all'inventario |
| RemoveObject(inventario, Object)->inventario | rimuove l'oggetto dall'inventario |

**Specifica Semantica:**

Decline: inventario:list , oggetto:Object

addObject(inventario(),oggetto)= inventario'

RemoveObject(inventario,oggetto)= inventario'

**Specifica di restrizione**

RemoveObject(Inventory())= errore

# 4.Applicazione argomenti:

## 4.1 Collection:

le collection implementate nel progetto sono:

**LIST**:

- La classe AdventureObjectConteiner implementa una lista dove sono presenti gli item che sono all'interno di altri item di tipo AdventureObjectConteiner.
- La classe Inventory implementa una lista di tipo AdventureObject dove sono presenti gli item che il giocatore può raccogliere.
- La classe Room implementa una lista degli item AdventureObject e un'altra lista di item AdventureObjectConteiner presenti in quella stanza

## 4.2 Eccezioni:

le eccezioni sono state implementate nel progetto per gestire vari tipo di errori presenti: errori di IO, errori di SQL ed altri tipi di errori.

Gli esempi di uso di eccezioni sono presenti negli argomenti successivi.

## 4.3 File

La gestione per la creazione e gestione dei file sono gestiti nella classe FileManager

```
public void update_savings_on_file_and_db(String username, String password, GameStatus saves){
    DBManager db = new DBManager();
    try {
        System.out.println("Creo il file");
        FileOutputStream fileOut = new FileOutputStream("src/main/resources/files/saves.dat");
        ObjectOutputStream outStream = new ObjectOutputStream(fileOut);
        outStream.writeObject(saves);

        FileInputStream fileIn = new FileInputStream("src/main/resources/files/saves.dat");
        db.update_savings_on_db(username, password, fileIn);

        fileOut.close();
        outStream.close();
        fileIn.close();
    } catch (IOException ex) {
        System.err.println(ex.getMessage());
    }
}
```

Questo metodo crea un nuovo oggetto di tipo db Manager, crea un nuovo file e lo inserisce nel db tramite il metodo **db.update_saving_on_db**.

```java
public GameStatus get_saves_from_file(String username, String password){
    DBManager db = new DBManager();
    GameStatus saves = null;
    if(Files.notExists(Paths.get("src/main/resources/files/saves.dat"))){
        try {
            Files.createFile(Paths.get("src/main/resources/files/saves.dat"));
            System.out.println("File creato con successo");
        } catch (IOException ex) {
            Logger.getLogger(FileManager.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    //prendo il file dal db
    db.get_saves_from_db(username, password);

    //deserializzo il file
    try (FileInputStream fileIn = new FileInputStream("src/main/resources/files/saves.dat");
         ObjectInputStream objectIn = new ObjectInputStream(fileIn)) {

        // Deserializzazione dell'oggetto GameStatus
        saves = (GameStatus) objectIn.readObject();
        System.out.println("\n\naaaaaaaaaaaaaaaa\n\n");
        // Stampa dell'oggetto deserializzato
        System.out.println("Oggetto GameStatus deserializzato: " + saves);

    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    // Stampa dell'oggetto deserializzato
    System.out.println("Oggetto GameStatus deserializzato: " + saves);

    try{
        FileOutputStream outFile = new FileOutputStream("src/main/resources/files/saves.dat");
        ObjectOutputStream outStream = new ObjectOutputStream(outFile);
        outStream.writeObject(saves);
    } catch (IOException ex) {
        System.err.println(ex.getMessage());
    }
    return saves;
}
```

Metodo di tipo **GameStatus** che serve per prendere i salvataggi dal db e caricarli in locale.

## 4.4 Database:

il database in questo progetto è stato implementato per gestire i salvataggi del gioco in base all'utente che accede al gioco tramite il **LoginFrame** mostrato ad ogni avvio del gioco.

La classe **DBmanager** serve per gestire il database H2 che abbiamo implementato nel progetto.

```
public class DBManager {

    final String CREATE_TABLE = "CREATE TABLE IF NOT EXISTS adventure_user (email VARCHAR(50), username VARCHAR(35), password VARCHAR(20), savings BLOB, PRIMARY KEY (email))";

    /**
     *
     */
    public DBManager() {
        try{
            Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

            Statement stm = conn.createStatement();
            stm.executeUpdate(CREATE_TABLE);
            stm.close();
            conn.close();
        } catch(SQLException ex){
            System.err.println(ex.getSQLState() + ": " + ex.getMessage());
        }

    }
```

Viene prima fatto un try catch per connettersi al database e creare la prima tabella (in caso contrario arriverà un messaggio di errore).

```
public boolean is_user_existent(String username){
    boolean answer = false;

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE username=?");
        pstm.setString(1, username);
        ResultSet rs = pstm.executeQuery();

        while (rs.next()){
            answer = true;
        }
        pstm.close();
        conn.close();

    } catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }

    return answer;
}
```

Questo metodo serve per verificare che l'utente inserito nel login esista nel database. All'inizio viene settato il valore booleano che verrà restituito a false, dopodiché si effettuerà una connessione al database con una query per verificare che l'usermane inserito sia presente. Il risultato sarà un oggetto di tipo ResultSet che verrà controllato dal while, che setterà il valore booleano di answer a true se sarà presente almeno un elemento nel ResultSet altrimenti restituirà false.

```java
public boolean user_has_savings(String username, String password){
    boolean answer = false;

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE username=? AND password=?");
        pstm.setString(1, username);
        pstm.setString(2, password);
        ResultSet rs = pstm.executeQuery();

        while (rs.next()){
            if (rs.getObject("savings") != null)
                answer = true;
        }
        pstm.close();
        conn.close();

    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }

    return answer;
}
```

Questo metodo invece controlla se l'utente ha dei salvataggi o meno sempre con lo stesso modus operandi del metodo precedente per cercare l'utente e password inserite nel DB, l'unica differenza è che nel while c'è un piccolo if che controlla se l'utente ha dei salvataggi o meno.

```java
public void update_savings_on_db(String username, String password, FileInputStream file_to_save) {

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("UPDATE adventure_user SET savings=? WHERE username=? AND password=?");
        pstm.setBinaryStream(1, file_to_save);
        pstm.setString(2, username);
        pstm.setString(3, password);
        int rowsAffected = pstm.executeUpdate();
        pstm.close();
        conn.close();
        System.out.println("db aggiornato, righe aggiornate: " + rowsAffected);

    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }
}
```

Questo metodo serve per aggiornare il salvataggio sul database.

```java
public void get_saves_from_db(String username, String password){

    InputStream input = null;

    try{
        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");
        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE username=? AND password=?");
        pstm.setString(1, username);
        pstm.setString(2, password);
        ResultSet rs = pstm.executeQuery();

        if (rs.next()){
            //saves = (GameStatus) rs.getObject("savings");
            input = rs.getBinaryStream("savings");

            try (OutputStream outputStream = new FileOutputStream("src/main/resources/files/saves.dat")) {
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = input.read(buffer)) != -1) {
                    outputStream.write(buffer, 0, bytesRead);
                }
            }
        }

        System.out.println("File salvato con successo");

        pstm.close();
        conn.close();
    } catch(IOException | SQLException ex){
        System.err.println(ex.getMessage());
    }
    System.out.print("\nretun inpu stream\n");
}
```

Questo metodo serve per recuperare il file dal database


```java
public void add_new_user(String email, String username, String password){
    try {
        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");
        PreparedStatement pstm = conn.prepareStatement("INSERT INTO adventure_user (email, username, password, savings) VALUES (?, ?, ?, NULL)");
        pstm.setString(1, email);
        pstm.setString(2, username);
        pstm.setString(3, password);


        int rowsAffected = pstm.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Nuovo utente inserito con successo!");
        }
        pstm.close();
        conn.close();

    } catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }
}
```

Metodo per aggiungere un nuovo utente nel DB

```java
public boolean username_already_exists(String username_to_check){
    boolean answer = false;

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE username=?");
        pstm.setString(1, username_to_check);
        ResultSet rs = pstm.executeQuery();

        while (rs.next()){
            if (username_to_check.compareTo(rs.getString("username"))==0)
                answer = true;
        }
        pstm.close();
        conn.close();

    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }

    return answer;
}
```

metodo per verificare se l'utente inserito nel login esista già nel DB. Dopo aver recuperato il ResultSet viene fatto un controllo con l'username tramite un if che compie una comparazione tra stringhe username (quello inserito) e quello presente nel rs , se esce 0 significa che le due stringhe sono uguali e quindi restituisce il valore true altrimenti rimane false.

```java
public boolean email_already_exists(String email_to_check){
    boolean answer = false;

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE email=?");
        pstm.setString(1, email_to_check);
        ResultSet rs = pstm.executeQuery();

        while (rs.next()){
            if (email_to_check.compareTo(rs.getString("email"))==0)
                answer = true;
        }
        pstm.close();
        conn.close();

    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }

    return answer;
}
```

Metodo per verificare che la mail inserita esista già nel DB. Simile alla precedente ma controlla l'e-mail non l'username.

```java
public boolean is_password_correct(String username, String password){
        boolean answer = false;

    try{

        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("SELECT * FROM adventure_user "
                + "WHERE username=?");
        pstm.setString(1, username);
        ResultSet rs = pstm.executeQuery();

        while (rs.next()){
            if (password.compareTo(rs.getString("password"))==0)
                answer = true;
        }
        pstm.close();
        conn.close();

    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }

    return answer;
}

/**
 * Metodo che elimina i salvataggi di un utente il cui username è passata per parametro.
 * @param username
 */
public void destroy_savings(String username){

    try{
        Connection conn = DriverManager.getConnection("jdbc:h2:./resources/db");

        PreparedStatement pstm = conn.prepareStatement("UPDATE adventure_user SET savings=null WHERE username=?");
        pstm.setString(1, username);
        int rowsAffected = pstm.executeUpdate();
        System.out.println("DB: "+username);
        pstm.close();
        conn.close();
    }catch(SQLException ex){
        System.err.println(ex.getSQLState() + ": " + ex.getMessage());
    }
}
```

Il metodo **is_password_correct** verifica se la password inserita combacia con l'username inserito. Stessa metodologia del metodo precedente ma esegue un controllo sulla password.

Il metodo **destroy_savings** distrugge i salvataggi dell'utente quando il timer arriva a zero. Si connette al DB e cerca l'utente che stava giocando in quel momento settando la sua colonna del salvataggio a NULL (quindi eliminandolo).

## 4.5 Threads:

I threads in questo progetto servono a gestire il timer di gioco che indica il tempo rimanente per vincere.

```java
public class Timer implements Serializable{

    private final int TIMER_VALUE = 1200;    // 1200s = 20min
    private int current_value;               // valore corrente del timer

    /**
     *
     */
    public Timer() {
        this.current_value = TIMER_VALUE;
    }

    /**
     *
     * @param current_value
     */
    public Timer(int current_value){
        this.current_value = current_value;
    }

    /**
     *
     * @return
     */
    public int getCurrent_value() {
        return current_value;
    }

    /**
     *
     * @param current_value
     */
    public void setCurrent_value(int current_value) {
        this.current_value = current_value;
    }

}
```

La classe Timer semplicemente inizializza il timer al valore di default(1200 sec= 20 min) oltre a inizializzare i metodi di get e set per prendere e settare il valore del timer stesso.

```java
public class TimerManager implements Runnable{

    private Timer timer;
    private boolean running;            // Flag per indicare se il timer è attivo
    private AdventureFrame adventureFrame;

    /**
     * Costruttore della classe TimerManager
     * @param timer
     */
    public TimerManager(Timer timer, AdventureFrame adventureFrame) {
        this.timer = timer;
        this.adventureFrame = adventureFrame;
        this.running = false;
    }

    /**
     * Metodo per fermare il timer
     */
    public void stopTimer() {
        running = false;
    }
```

```java
    @Override
    public void run() {
        running = true;
        while (running && timer.getCurrent_value() > 0) {
            try {
                Thread.sleep(1000); // Pausa di 1 secondo
                timer.setCurrent_value(timer.getCurrent_value() - 1); // Decremento di currentValue

                int minutes = timer.getCurrent_value() / 60;
                int seconds = timer.getCurrent_value() % 60;
                String timeString = String.format("%02d:%02d", minutes, seconds);

                adventureFrame.setTimerLabel(timeString);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("Il thread del timer è stato interrotto.");
            }
        }
        if (timer.getCurrent_value() <= 0) {
            adventureFrame.game_lost();
            System.out.println("TEMPO SCADUTO!.");
        }
        running = false;
    }
}
```

Il timer manager invece serve per gestire il timer durante la partita. Il timer parte quando inizia la partita e man mano viene decrementato di un secondo tramite **timer.setCurrent_value** che prende il valore iniziale del timer e lo decrementa di 1.

Il minutaggio viene stampato a schermo con una stringa in minuti e secondi.

Se il timer durante la partita arriva a 0 viene stampato un messaggio:” tempo scaduto” e dopo viene fermato.

## 4.6 Socket:

l'utilizzo dei socket è stato implementato per la creazione e gestione di una chat tra utenti.

```
public class ClientFrame extends javax.swing.JFrame {

    private Users user;
    private PrintWriter out;
    private static final String SERVER_ADDRESS = "127.0.0.1"; // Indirizzo del server
    private static final int SERVER_PORT = 12345;
    private Socket socket;

    /**
     * Creates new form ClientFrame
     */
    public ClientFrame(Users user_info) {
        this.user = user_info;
        initComponents();
        setLocationRelativeTo(null);
    }
```

La classe ClientFrame è usata per gestire la parte Client.

```
    private void closeConnection() {
        try {
            if (socket != null && !socket.isClosed()) {
                socket.close();
            }
            dispose();
            System.out.println("Connessione chiusa e applicazione terminata.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Metodo usato per chiudere la connessione.

```java
private void rispondi_button1ActionPerformed(java.awt.event.ActionEvent evt) {

    String user_I_am = user.getUsername();

    new Thread(() -> {
        try {
            System.out.println("Tentativo di connessione al server " + SERVER_ADDRESS + " sulla porta " + SERVER_PORT + "...");
            socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
            System.out.println("Chiamata avviata!");
            label_messaggio.setForeground(Color.green);
            label_messaggio.setText("Chiamata avviata!! Parla col tuo amico.");

            inputArea.setEnabled(true);
            send_button.setEnabled(true);

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);

            // Invio del nome del client al server
            out.println(user_I_am);

            String serverResponse;
            if ((serverResponse = in.readLine()) != null) {
                System.out.println(serverResponse);
                messageArea.append(serverResponse + "\n");
                if (serverResponse.equals("Nome utente non autorizzato.")) {
                    System.out.println("Connessione chiusa dal server.");
                    label_messaggio.setForeground(Color.red);
                    label_messaggio.setText("Nessuno ti sta chiamando. Che tristezza...");
                    socket.close();
                    return;
                }
```

Metodo usato se si riceve una chiamata da un altro utente. Viene creato un nuovo thread per gestire la chiamata.

Verifica se arriva una risposta dal Server(verifica se qualcuno lo sta chiamando)

```java
                Thread readerThread = new Thread(() -> {
                    try {
                        String serverMessage;
                        while ((serverMessage = in.readLine()) != null) {
                            messageArea.append(serverMessage + "\n");
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                });
                readerThread.start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();
    }
```

C’è anche un thread che gestisce la ricezione del messaggio.

```java
public class ServerFrame extends javax.swing.JFrame {

    private PrintWriter out;
    private static final int PORT = 12345; // Porta del server
    private Users user = null;
    private Socket clientSocket;
    private ServerSocket serverSocket;

    /**
     * Creates new form chatFrame
     */
    public ServerFrame(Users user_info) {
        this.user = user_info;
        initComponents();
        setLocationRelativeTo(null);
    }
```

La classe ServerFrame serve per gestire la parte server della chiamata.

```java
    private void closeConnection() {
        try {
            if (clientSocket != null && !clientSocket.isClosed()) {
                clientSocket.close();
            }
            if (serverSocket != null && !serverSocket.isClosed()) {
                serverSocket.close();
            }
            dispose();
            System.out.println("Connessione chiusa e applicazione terminata.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Metodo per chiudere la connessione.

```
private void chiama_buttonActionPerformed(java.awt.event.ActionEvent evt) {
    //rispondi_button.setEnabled(false);
    if (email_field.getText() != null && !email_field.getText().isBlank() && !email_field.getText().isEmpty()){
        String user_I_want = email_field.getText();

        DBManager db = new DBManager();

        if (db.is_user_existent(user_I_want)){
            if (!user.getUsername().equals(user_I_want)){
                System.out.println("Server avviato...");
                label_messaggio.setForeground(Color.green);
                label_messaggio.setText("Chiamata avviata verso " + user_I_want + ", in attesa di risposta...");
                new Thread(() -> {
                    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
                        System.out.println("In attesa di connessioni sulla porta " + PORT + "...");
                        while (true) {
                            clientSocket = serverSocket.accept();

                            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                            out = new PrintWriter(clientSocket.getOutputStream(), true);

                            String clientName = in.readLine();
                            System.out.println("Nome client ricevuto: " + clientName);

                            if (user_I_want.equals(clientName)) {
                                label_messaggio.setForeground(Color.green);
                                label_messaggio.setText("Utente connesso alla chiamata! Parla col tuo amico.\n");

                                inputArea.setEnabled(true);
                                send_button.setEnabled(true);

                                Thread readerThread = new Thread(() -> {
                                    try {
                                        String inputLine;
                                        while ((inputLine = in.readLine()) != null) {
                                            messageArea.append(inputLine + "\n");
                                        }
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                    }
                                });
                                readerThread.start();
                            } else {
                                messageArea.append("Client non autorizzato. Connessione chiusa.\n");
                                out.println("Nome utente non autorizzato.");
```

Questo metodo entra in azione dopo che l'utente ha premuto il pulsante chiama.

Verifica se la mail inserita dall'utente è scritta in modo consono, utilizzando una *regex*, poi verifica se la mail e l'username collegato alla mail è presente nel db. Avviato il server viene creato un nuovo thread che gestisce la chiamata tra i due utenti (dopo aver verificato il collegamento).

```
                                out.println("Nome utente non autorizzato.");
                                clientSocket.close();
                            }
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }).start();
            } else {
                label_messaggio.setForeground(Color.red);
                label_messaggio.setText("Chiamare se stessi è troppo triste,\nnon posso lasciartelo fare...");
            }
        } else {
            label_messaggio.setForeground(Color.red);
            label_messaggio.setText("Non esiste nessun utente con questo username!!");
        }
    } else {
        label_messaggio.setForeground(Color.red);
        label_messaggio.setText("Compilare il campo richiesto!!");
    }
}
```

Questi sono messaggi in caso in cui una condizione dei vari if non venga rispettata.

```
private void send_buttonActionPerformed(java.awt.event.ActionEvent evt) {
    if (!inputArea.getText().isBlank() && !inputArea.getText().isEmpty()){
        if (out != null) {
            out.println("[" + user.getUsername() + "]: " + inputArea.getText());
            messageArea.append("[" + user.getUsername() + "]: " + inputArea.getText() + "\n");
            inputArea.setText("");
        }
    }
}

/**
 *
 * @param evt
 */
private void closeConn_buttonActionPerformed(java.awt.event.ActionEvent evt) {
    closeConnection();
}

/**
 *
 */
public void runServerFrame() {

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ServerFrame(user).setVisible(true);
            System.out.println(user.getUsername());
            inputArea.setEnabled(false);
            send_button.setEnabled(false);
        }
    });

}
```

Oltre al metodo runServerFrame per avviare il frame dedicato al server ed al metodo del bottone per chiudere la connesione (closeConn_buttonActionPerformed) c'è anche il metodo per il bottone quando si vuole inviare un messaggio all'utente con cui si sta chattando.

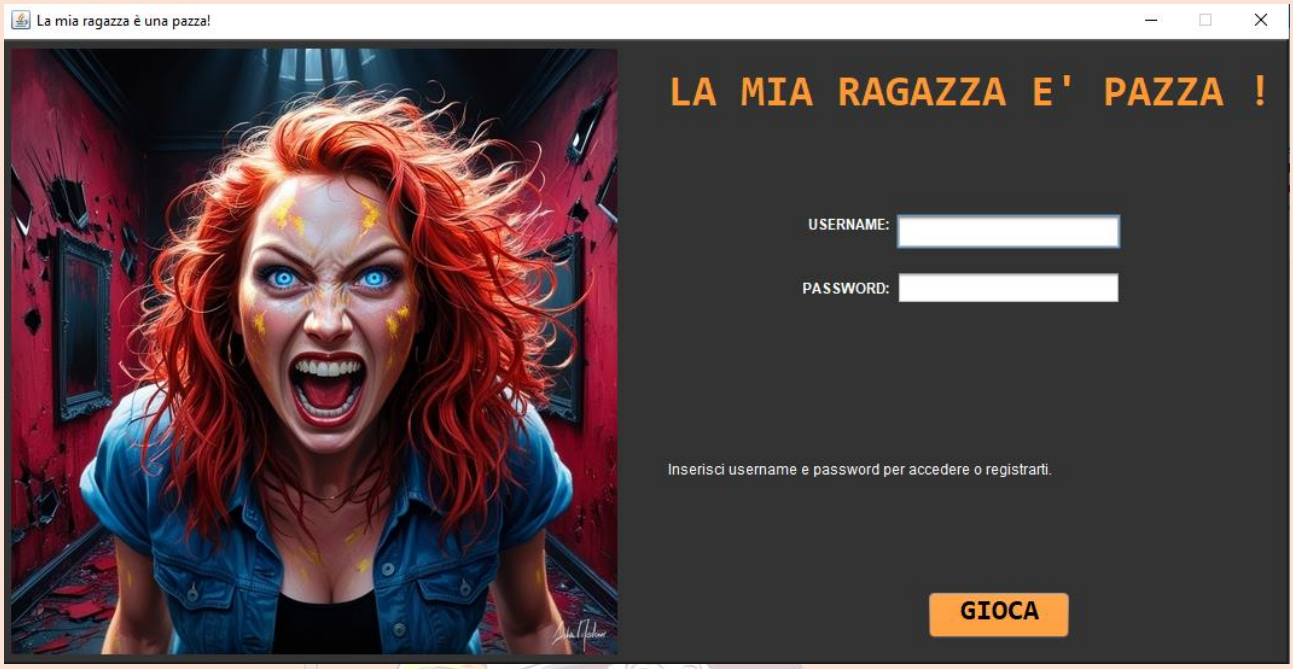
## 4.7 lambda expressions:

sono state implementate nell"AdvenutureFrame per effettuare dei cicli(esempio cicli di lettura) all'interno delle varie liste presenti nel codice.

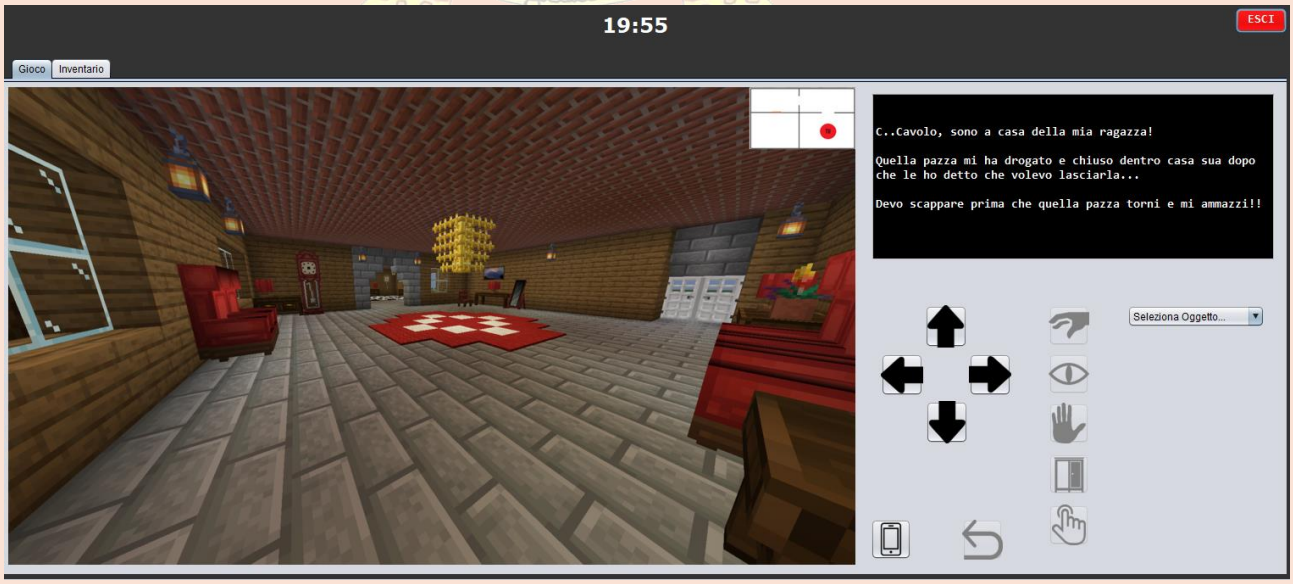
```
public void aggiornaOggettiStanza(){
    itemDisplayer_comboBox.removeAllItems();  //cancello item precedenti
    itemDisplayer_comboBox.addItem("Seleziona Oggetto..."); //aggiungo item "fantoccio"
    game_status.getCurrent_room().getObjects().forEach((itemStanza) -> { //scorro sia le liste di item che containers nella stanza
        if(!"Pezzo di chiave #2".equals(itemStanza.getName())){
            itemDisplayer_comboBox.addItem(itemStanza.getName());                    //e ne inserisco il nome della comboBox
        }
    });
    game_status.getCurrent_room().getContainers().forEach((containerStanza) -> {
        itemDisplayer_comboBox.addItem(containerStanza.getName());
    });

    //disabilito inizialmente i bottoni di interazione
    pickUp_button.setEnabled(false);
    move_button.setEnabled(false);
    observe_button.setEnabled(false);
    open_button.setEnabled(false);
    interaction_button.setEnabled(false);
    backToRoom_button.setEnabled(false);
    up_button.setEnabled(true);
    down_button.setEnabled(true);
    right_button.setEnabled(true);
    left_button.setEnabled(true);
}

//Metodo come il precedente, visualizza nella comboBox della stanza gli item di un contenitore selezionato
public void aggiornaOggettiContainer(AdventureObjectContainer containerOggetti){
    itemDisplayer_comboBox.removeAllItems();  //cancello item precedenti
    itemDisplayer_comboBox.addItem("Seleziona Oggetto..."); //aggiungo item "fantoccio"
    containerOggetti.getList().forEach((itemContainer) -> { //scorro la lista degli item del container
        itemDisplayer_comboBox.addItem(itemContainer.getName());
    });
}

//Metodo come il precedente, visualizza nella comboBox dell'inventario gli item che possiede il giocatore
public void aggiornaOggettiInventario(){
    itemComboBox.removeAllItems();  //cancello item precedenti
    itemComboBox.addItem("Seleziona Oggetto..."); //aggiungo item "fantoccio"
    game_status.getInventario().getList().forEach((itemInventario) -> { //scorro la lista degli item del'iventario
        itemComboBox.addItem(itemInventario.getName());
    });
```

## 4.8 Swing:

Le swing sono state implementate per la parte grafica del gioco.

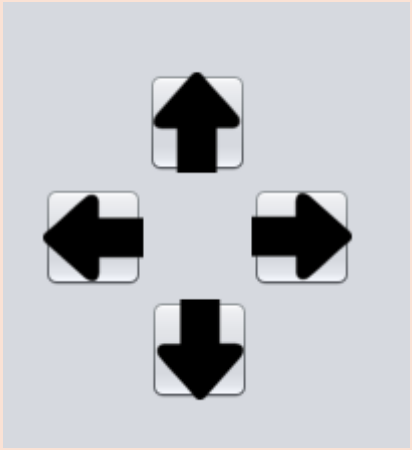



Questa è la schermata di login dove vengono inserite dall'utente username e password che poi vengono controllate e verificate se sono presenti nel database. In caso non siano presenti si crea un nuovo utente con quelle credenziali e poi dopo aver effettuato il login si può avviare la partita e verrà creato anche un nuovo file di salvataggio

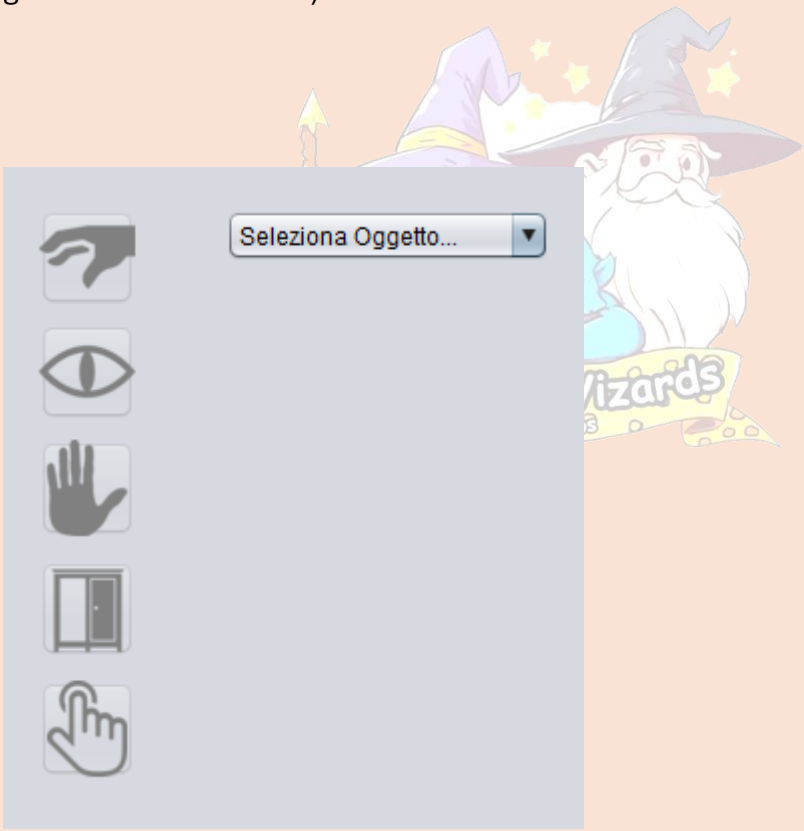



questa è la schermata dopo aver effettuato il login. Ci sono due tab in questo JPanel: uno è usato per gestire l'interfaccia di gioco mentre l'altro per l'inventario. Il gioco è gestito in questo modo: a sinistra è presente l'immagine della stanza in cui ci si trova

mentre a destra ci sono varie azioni eseguibili e una finestra di dialogo dove sono presenti sia dialoghi di gioco che descrizione delle stanze.

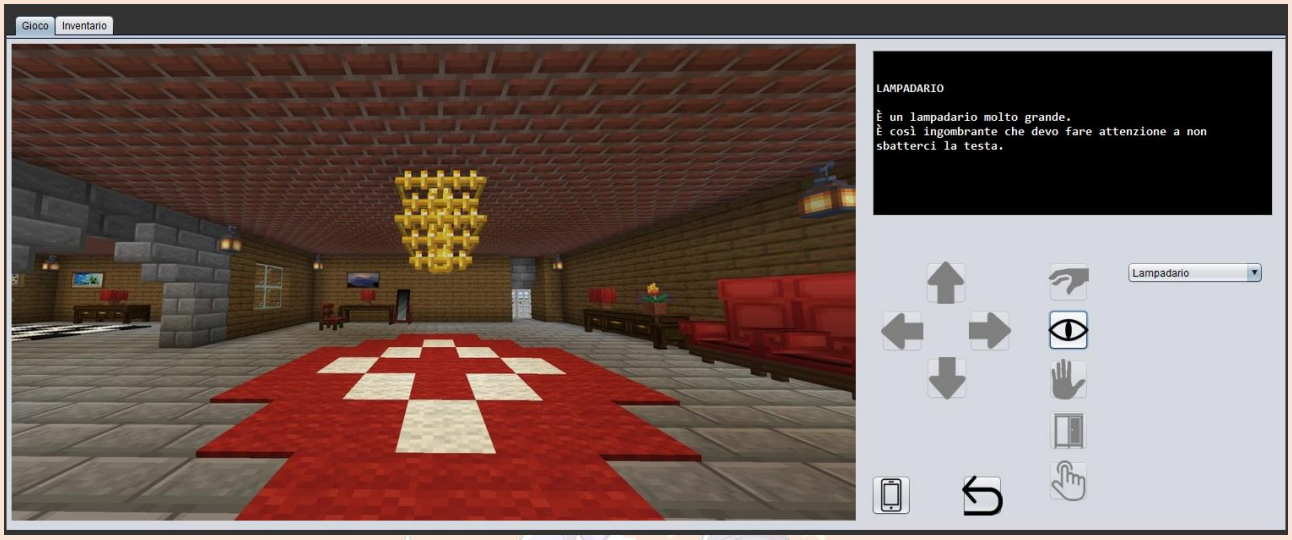


Ci sono dei JBottom che servono come tasti direzionali per muoversi nelle varie stanze (è stata inserita in ogni immagine delle stanze una mini-mappa per far capire al giocatore dove si trova).




Questa parte del JPanel serve a gestire gli oggetti presenti nella stanza. Con la itemDisplayer_comboBox è possibile visionare tutti gli item nella stanza e non appena lo si clicchi l'immagine della stanza viene sostituita con quella dell'oggetto in questione con cui è possibile interagire con i seguenti tasti descritti dal più alto al più basso: Prendi, Osserva, Muovi, Apri, Usa. Ogni tasto verrà evidenziato in base agli attributi dell'oggetto.

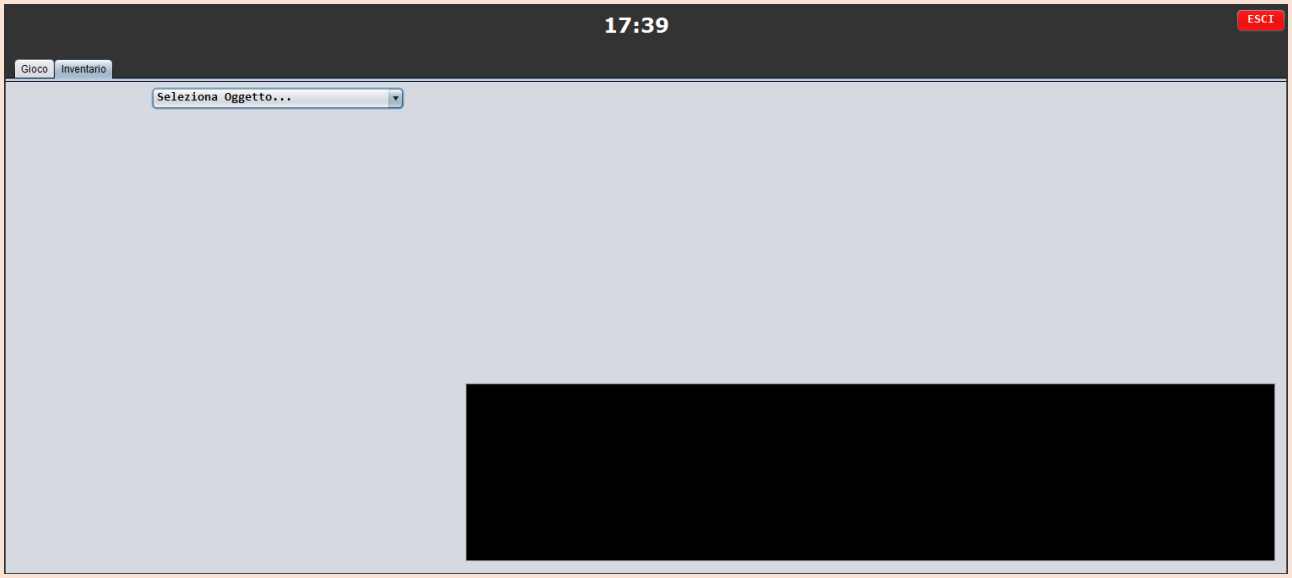
Esempio:




In questo esempio ho selezionato l'oggetto lampadario e dato che tale oggetto è solo osservabile si potrà utilizzare solo il tasto relativo che farà visualizzare la sua descrizione nella finestra di dialogo.

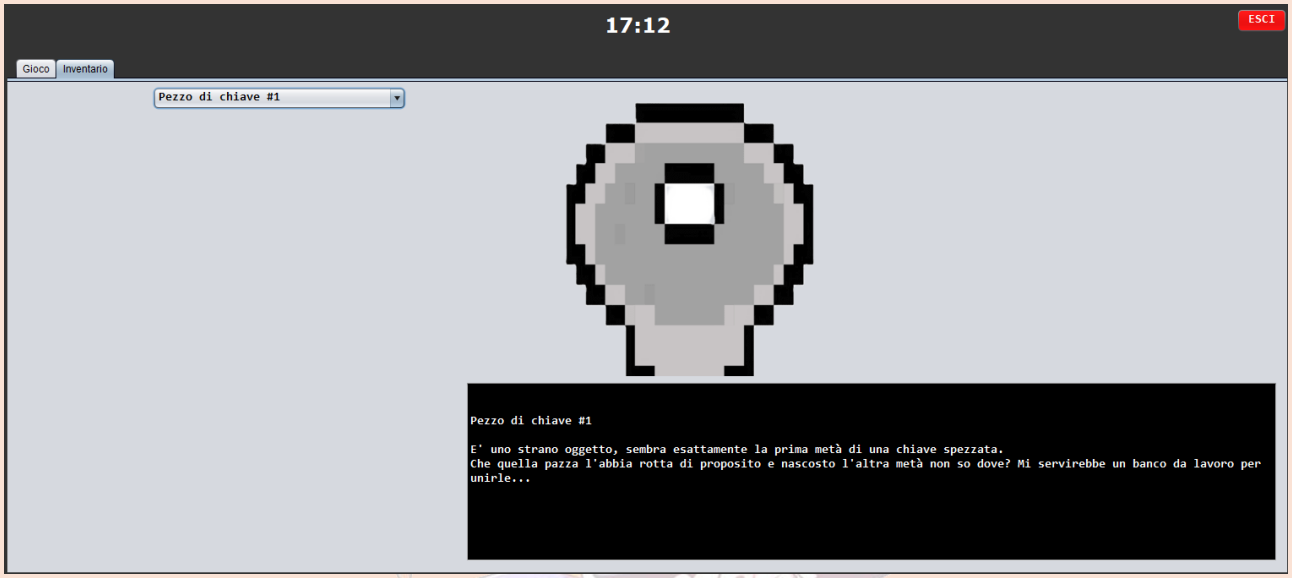
Per tornare a visualizzare la stanza basta premere il bottone con la freccia rivolta verso sinistra.

L'inventario nella seconda tab si presenta così:



È presente una ComboBox dove è possibile selezionare gli oggetti raccolti in giro per la casa e non appena l'oggetto desiderato sarà selezionato apparirà un'immagine dell'oggetto in questione con sotto una sua descrizione.

Ecco un esempio:



## 5.Guida per finire il gioco:

- **Passo 1:** Si parte dalla stanza soggiorno e selezionare l'oggetto "comodino" nella ComboBox e premere il tasto apri, dopodiché selezionare nella stessa comboBox il pezzo di chiave #1 e premere il tasto prendi per inserirlo nell'inventario. Premere il tasto della freccia verso sinistra per tornare nella stanza soggiorno.
- **Passo 2:** Andare nella stanza da lavoro (premendo la freccia su), selezionare il tappetto a scacchiera e premere il tasto muovi per recuperare il Pezzo di chiave #2. Premere il tasto della freccia verso sinistra per tornare nella stanza da lavoro.
- **Passo 3:** selezionare l'oggetto banco da lavoro e premere il tasto usa e si creerà la chiave del giardino. Premere il tasto della freccia verso sinistra per tornare nella stanza da lavoro.
- **Passo 4:** andare nella stanza cucina (premendo la freccia a sinistra) dopodiché selezionare l'oggetto porta e premere il tasto apri per aprirla così da poter accedere in giardino. Premere il tasto della freccia verso sinistra per tornare nella stanza cucina.
- **Passo 5:** andare nella stanza Giardino (premendo la freccia giù), selezionare l'oggetto armadio degli oggetti, premere il tasto apri e selezionare l'oggetto ascia da boscaiolo, per poi prendere con il tasto prendi. Premere il tasto della freccia verso sinistra per tornare nella stanza Giardino.
- **Passo 6:** tornare in cucina (premere la freccia su), selezionare dispensa della cucina e premere il tasto apri, poi selezionare osso e premere il tasto prendi, Premere il tasto della freccia verso sinistra per tornare nella stanza cucina.
- **Passo 7:** tornare in giardino (premere il tasto giù) e selezionare l'oggetto cuccia del cane, premere il tasto usa e poi il tasto apri e selezionare chiave d'oro e premere il tasto prendi. Premere il tasto della freccia verso sinistra per tornare nella stanza Giardino.
- **Passo 8:** premere la seguente serie di tasti per tornare in stanza soggiorno e vincere il gioco: freccia su, freccia destra, freccia giù.