



الكلية متعددة التخصصات النافور

ⵜⴰⵎⴻⵔⴰⵏⵜ ⵜⴰⵎⴻⵔⴰⵏⵜ ⵜⴰⵎⴻⵔⴰⵏⵜ | II.E.Q

Faculté Pluridisciplinaire de Nador

Mini Projet

ENCADRÉE PAR : EL MAKKAOUI KHALID.

RÉALISÉ PAR : NITA YOUSSEF.

Partie II :

« Programmation Orientée Objet . »

« Sujet : Jeu de Mémoire. »

Introduction:

Ce rapport est mise pour expliquer presque tous les nécessaire partie du système (partie programmation)

Par Utilisation du Langage de programmation java on a créé un jeu de mémoire essentiellement composé par login interface , settings interface et l'interface du jeu . Chacun de ces éléments a un élément secondaire attaché à lui comme par exemple:d'après Login Interface on peut accéder au registre interface, d'après settings dès dialogue de configuration

[I. Login Interface](#)

[II. Registre Interface:](#)

[III. Settings:](#)

[III. DataBase:](#)

[IV. Controller](#)

[V. Class Card :](#)

[VI. Class PlayGameWithSingle :](#)

[VII. ScoreUpdate](#)

[VIII. CustomTableCellRenderer](#)

I.Login Interface

1. Les Méthodes, Buttons et Leur Fonctionnement 🍌:

Vue general du Class **Login** :

```
import ...5 lines
public class login extends javax.swing.JFrame {
    /**
     * Creates new form login
     */
    public login() {...5 lines }
    /** This method is called from within the constructor to
    @SuppressWarnings("unchecked")
    Generated Code

    private void jLabel2MouseClicked(java.awt.event.MouseEvent
        System.exit( status:0);
    }

    private void disableMouseClicked(java.awt.event.MouseEvent
    private void showMouseClicked(java.awt.event.MouseEvent e
    private void formWindowOpened(java.awt.event.WindowEvent
    private void jButton1ActionPerformed(java.awt.event.Action
    private void jLabel13MouseClicked(java.awt.event.MouseEve

    public static void main(String args[]) {
```

Le code importe trois paquets Java :

- **java.sql**: fournit l'API pour accéder et traiter les données stockées dans une source de données (généralement une base de données relationnelle) à l'aide du langage de programmation Java.
- **java.sql.PreparedStatement**: fournit un moyen d'exécuter une instruction SQL précompilée avec ou sans paramètres.

- **javax.swing.JOptionPane**: fournit un moyen d'interagir avec l'utilisateur en utilisant une fenêtre contextuelle.

1.1. Public Login():

```
public login() {  
    initComponents();  
    txtusername.setBackground(new java.awt.Color( r:0, g:0, b:0, a:1));  
    txtpassword.setBackground(new java.awt.Color( r:0, g:0, b:0, a:1));  
}  
/** This method is called from within the constructor to initialize the form
```

La méthode **login** initialise les composants graphiques et définit la couleur de fond des champs de saisie **txtusername** et **txtpassword**. La couleur de fond est définie en créant un nouvel objet de couleur avec des valeurs RGB 0,0,0 et une opacité de 1.

1.2. private void jLabel2MouseClicked(java.awt.event.MouseEvent evt):

Cette méthode privée **jLabel2MouseClicked** permet de quitter l'application en cliquant sur le label **jLabel2**. Le paramètre **evt** représente l'événement de

```
private void jLabel2MouseClicked(java.awt.event.MouseEvent evt) {  
    System.exit( status:0);  
}
```

souris qui déclenche la méthode. La méthode appelle la méthode **System.exit** (0), ce qui indique à l'application de se terminer avec un code de sortie 0, qui signifie une sortie normale.

1.3. private void disableMouseClicked(java.awt.event.MouseEvent evt):

```
private void disableMouseClicked(java.awt.event.MouseEvent evt) {  
    txtpassword.setEchoChar((char)0);  
    disable.setVisible(aFlag: false);  
    disable.setEnabled(enabled: false);  
    show.setEnabled(enabled: true);  
    show.setEnabled(enabled: true);  
}
```

Cette méthode privée disableMouseClicked permet de désactiver le masquage du mot de passe dans un champ de saisie de mot de passe. Le paramètre evt représente l'événement de souris qui déclenche la méthode. La méthode définit le caractère d'écho du champ txtpassword en 0 pour ne plus masquer le mot de passe. Le bouton disable est ensuite rendu invisible et désactivé, tandis que le bouton show est activé et rendu visible.

1.4. private void showMouseClicked(java.awt.event.MouseEvent evt):

```
private void showMouseClicked(java.awt.event.MouseEvent evt) {  
    txtpassword.setEchoChar((char)8226);  
    disable.setVisible(aFlag: true);  
    disable.setEnabled(enabled: true);  
    show.setEnabled(enabled: false);  
    show.setEnabled(enabled: false);  
}
```

Cette méthode privée showMouseClicked permet d'afficher le masquage du mot de passe dans un champ de saisie de mot de passe. Le paramètre evt représente l'événement de souris qui déclenche la méthode. La méthode définit le caractère d'écho du champ txtpassword en 8226 pour masquer le mot de passe. Le bouton show est ensuite rendu invisible et désactivé, tandis que le bouton disable est activé et rendu visible.

1.5. private void formWindowOpened(java.awt.event.WindowEvent evt):

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {  
    for (double i = 0.0; i <=1.0; i = i+0.1){  
        String val = i+ "";  
        float f = Float.parseFloat(s:val);  
        this.setOpacity( opacity:f);  
        try{  
            Thread.sleep( millis: 50);  
        }catch(InterruptedException e){  
        }  
    }  
}
```

Cette méthode privée s'exécute lorsqu'une fenêtre est ouverte. Elle effectue une boucle en incrémentant de 0,1 à chaque tour jusqu'à ce que la valeur soit égale à 1,0. Pour chaque tour, la chaîne "val" est définie comme étant égale à "i", puis convertie en un nombre flottant "f". Ensuite, la méthode "setOpacity" est appelée pour définir l'opacité de la fenêtre à la valeur de "f". Enfin, le thread en cours est mis en sommeil pendant 50 millisecondes.

1.6. private void jButton1ActionPerformed(java.awt.event.ActionEvent evt):

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    String username = txtusername.getText();// get the entered username from the text field  
    String password = new String( value:txtpassword.getPassword());// get the entered password f  
    try { // Connect to the database  
        Connection con = DriverManager.getConnection( url:"jdbc:mysql://localhost:3306/miniprojet?ze  
            user:"root", password:"090909");  
        // Create a statement to execute the query  
        PreparedStatement stmt = con.prepareStatement( string:"SELECT * FROM accounts WHERE username  
        stmt.setString( i:1, string:username);  
        stmt.setString( i:2, string:password);  
        // Execute the query to check if the username and password match  
        ResultSet rs = stmt.executeQuery();  
        // Check if a match was found  
        if (rs.next()) {  
            this.dispose();  
            InitialScreen theView = new InitialScreen();  
            //view.initLaunchScreen();  
            Settings theModel = new Settings();  
            //InitialScreen theView = new InitialScreen();  
            // theModel = theView.gameParams;  
            //System.out.println(theModel.getSingleName());  
            //Controller theController = new Controller(theView, theModel);  
            Controller theController = new Controller(theView);  
            theController.controllerStartInitialScreen();  
        } else {  
            JOptionPane.showMessageDialog( parentComponent:null, message:"Invalid username or password. Pleas  
                title:"Error", messageType:JOptionPane.ERROR_MESSAGE);  
        }  
    } catch (SQLException e) {}  
}
```

Ce code représente une méthode en Java qui est appelée lorsque le bouton "jButton1" est cliqué. La méthode effectue les opérations suivantes:

- Récupère le nom d'utilisateur et le mot de passe entré dans les champs de texte correspondants.
- Établit une connexion à la base de données MySQL en utilisant les informations de connexion (URL de la base de données, nom d'utilisateur et mot de passe).
- Prépare une requête SQL pour vérifier si les informations d'identification correspondantes sont présentes dans la table "accounts".
- Exécute la requête et vérifie si des résultats sont trouvés.
- Si des résultats sont trouvés, la fenêtre actuelle est fermée et une nouvelle fenêtre "InitialScreen" est ouverte. En même temps, un nouvel objet "Controller" est créé pour contrôler la vue et le modèle associé.
- Si aucun résultat n'est trouvé, une boîte de dialogue est affichée pour indiquer que le nom d'utilisateur ou le mot de passe est incorrect.

1.7. `private void jLabel13MouseClicked(java.awt.event.MouseEvent evt):`

```
private void jLabel13MouseClicked(java.awt.event.MouseEvent evt) {  
    RegisterForm registre = new RegisterForm();  
    registre.setVisible( b:true);  
    dispose();  
}
```

Cette méthode en Java affiche une nouvelle fenêtre de formulaire d'inscription lorsqu'un utilisateur clique sur le label JLabel13. Elle ferme également la fenêtre actuelle.

1.8. `Main():`

```
public static void main(String args[]) {  
    /* Set the Nimbus look and feel */  
    Look and feel setting code (optional)  
  
    /* Create and display the form */  
    java.awt.EventQueue.invokeLater(() -> {  
        new login().setVisible( b:true);  
    });  
}
```

Ceci est la méthode principale du programme Java. Le programme tente de définir l'apparence et le style de son interface graphique utilisateur (GUI) sur Nimbus (un style de GUI intégré dans Java SE 6). Si l'apparence et le style Nimbus n'est pas disponible, il reste avec l'apparence et le style par défaut. La structure try-catch est utilisée pour gérer les exceptions qui peuvent survenir lors de la définition de l'apparence et du style. Enfin, le programme crée et affiche un formulaire en créant une instance de la classe "login" et en définissant sa visibilité sur "vrai".

II. Registre Interface:

1. Vue generale du class:

```
import ...4 lines
public class RegisterForm extends JFrame {
    private JLabel nameLabel, passwordLabel;
    private JTextField nameField;
    private JPasswordField passwordField;
    private JButton registerButton;
    public RegisterForm() {...60 lines }
    public static void main(String[] args) {...6 lines }
}
```

2. L'explication de la class :

Cette classe Java représente un formulaire d'inscription pour une application. Voici les étapes principales effectuées dans ce code :

- Importation des bibliothèques nécessaires (java.awt, java.awt.event, javax.swing, java.sql)
- Création de la classe RegisterForm qui étend la classe JFrame.
- Définition des différents composants graphiques du formulaire, tels que les étiquettes de nom d'utilisateur et de mot de passe, les champs de texte pour le nom d'utilisateur et le mot de passe, et le bouton d'enregistrement.
- Configuration de la disposition du formulaire en utilisant un GridBagLayout.
- Association d'un ActionListener au bouton d'enregistrement pour exécuter une action lorsque le bouton est cliqué. L'action consiste à récupérer le nom d'utilisateur et le mot de passe entrés par l'utilisateur, puis à se connecter à la base de données et à insérer ces informations.
- Configuration de la taille et de l'emplacement du formulaire en utilisant les méthodes setSize et setLocationRelativeTo.
- Le mot de passe est hashé et salé en utilisant la fonction getHashedPassword() et la fonction getSalt(). La fonction getSalt() génère un sel aléatoire qui sera utilisé pour saler le mot de passe, et la fonction getHashedPassword() applique un algorithme de hashage

(PBKDF2WithHmacSHA1) pour stocker le mot de passe de manière sécurisée dans la base de données.

- La connexion à la base de données est établie en utilisant la classe Connection. Une instruction préparée est créée pour insérer les informations de l'utilisateur (nom, mot de passe hashé et sel) dans la table "accounts" de la base de données.
- Si l'insertion est réussie, une boîte de dialogue est affichée pour indiquer que l'enregistrement a réussi et les champs de nom et de mot de passe sont effacés. Si une exception est levée, une autre boîte de dialogue est affichée pour indiquer que l'enregistrement a échoué.
- Ajout d'un WindowListener pour afficher l'interface de connexion lorsque le formulaire est fermé.
- Définition de la méthode main qui crée une nouvelle instance du formulaire et la rend visible à l'utilisateur.

III. Settings:

1. Vue Generale du Class:

```
import ...4 lines
public class Settings { // Define getters and setters to update game parameters
    private int colId = 4;
    private int rowId = 3;
    private int diffLevel = 1000;
    private int timeInfo = 60;
    private String playerName;
    private String cardTheme = "socialmedia";
    private Color backgroundColor = Color.white;
    private boolean singlePlayer;
    private List<String> multiplePlayerName = Arrays.asList(a: "Player 1 ");
    public int getDiffLevel() {...5 lines }
    public void setDiffLevel(int level) {...6 lines }
    public String getCardTheme() {...5 lines }
    public void setCardTheme(String theme) {...6 lines }
    public Color getBackColor() {...5 lines }
    public void setBackColor(Color color) {...6 lines }
    public void setrowId(int id) {...3 lines }
    public int getrowId() {...3 lines }
    public void setcolId(int id) {...4 lines }
    public int getcolId() {...3 lines }
    public int getTimeInfo() {...5 lines }
    public void setTimeInfo(int time) {...4 lines }
    public String setSingleName(String name) {...4 lines }
    public boolean setSinglePlayer(boolean name) {...8 lines }
    public boolean getSinglePlayer() {...5 lines }
    public String getSingleName() {...3 lines }
    public List<String> getPlayersName() {...3 lines }
    public void setPlayersName(List<String> tmpList) {...3 lines }
```

Le code importe trois different package:

- java.awt.* - Ce package fournit des classes pour le développement d'interfaces graphiques pour les applications Java.
- java.util.* - Ce package contient des classes g n rales pour les op rations de collecte de donn es telles que les tableaux et les collections.
- java.util.List - Ce package sp cifique   la classe contient une interface utilis e pour les collections d'objets qui peuvent  tre index es et modifi es.

2. Les Attributs de la class Setting:

```
public class Settings { // Define getters and setters to update game parameters.
    private int colId = 4;
    private int rowId = 3;
    private int diffLevel = 1000;
    private int timeInfo = 60;
    private String playerName;
    private String cardTheme = "classes";
    private Color backgroundColor = Color.white;
    private boolean singlePlayer;
    private List<String> multiplePlayerName = Arrays.asList(a: "Player 1 ");
}
```

Ces attributs sont utiles dans getters et setters de la Class **Setting**

3. Les getters et setters de la class :

```
public int getDiffLevel() {
    return this.diffLevel;
}
public void setDiffLevel(int level) {
    this.diffLevel = level;
    System.out.println("Setting is ok " + level);
}
```

Les méthodes "**getDiffLevel()**" et "**setDiffLevel(int level)**" permettent de lire et de définir la valeur de la variable "**diffLevel**".

```
public String getCardTheme() {
    return this.cardTheme;
}
public void setCardTheme(String theme) {
    this.cardTheme = theme;
    System.out.println("Setting is ok " + theme);
}
```

Les méthodes "getCardTheme()" et "setCardTheme(String theme)" permettent de lire et de définir la valeur de la variable "cardTheme".

```
public Color getBackColor() {  
    return this.backgroundColor;  
}  
public void setBackColor(Color color) {  
    this.backgroundColor = color;  
    System.out.println("Setting is ok " + color);  
}
```

Les méthodes "getBackColor()" et "setBackColor(Color color)" permettent de lire et de définir la valeur de la variable "backgroundColor".

```
public void setrowId(int id) {  
    this.rowId = id;  
}  
public int getrowId() {  
    return this.rowId;  
}
```

Les méthodes "getrowId()" et "setrowId(int id)" permettent de lire et de définir la valeur de la variable "rowId".

```

public int getTimeInfo() {

    return this.timeInfo;
}
public void setTimeInfo(int time) {

    this.timeInfo = time;
}

```

Les méthodes "getTimeInfo()" et "setTimeInfo(int time)" permettent de lire et de définir la valeur de la variable "timeInfo".

```

public String setSingleName(String name) {
    return this.playerName = name;
}
public boolean setSinglePlayer(boolean name) {

    if (name) {
        return this.singlePlayer = true;
    } else {
        return this.singlePlayer = false;
    }
}
public boolean getSinglePlayer() {
    return this.singlePlayer;
}
public String getSingleName() {
    return this.playerName;
}
public List<String> getPlayersName() {
    return multiplePlayerName;
}
public void setPlayersName(List<String> tmpList) {
    this.multiplePlayerName = tmpList;
}

```

- **setSingleName(String name)** - définit l'attribut **playerName** à la valeur passée en argument.
- **setSinglePlayer(boolean name)** - définit l'attribut **singlePlayer** soit à **true** soit à **false** en fonction de la valeur booléenne passée en argument.
- **getSinglePlayer()** - retourne l'attribut **singlePlayer**.
- **getSingleName()** - retourne l'attribut **playerName**.
- **getPlayersName()** - retourne l'attribut **multiplePlayerName**.
- **setPlayersName(List<String> tmpList)** - définit l'attribut **multiplePlayerName** à la valeur passée en argument.

III. DataBase:

1. Vue Générale Du code:

```
public class Database {
    private static final String URL = "jdbc:mysql://localhost:3306/miniprojet?zeroDate";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "090909";
    public static Connection getConnection() { ...8 lines }
    public static String retrieveWordFromDatabase(String cardTheme) { ...12 lines };
    public static String retrieveWordFromDatabase(int no,String cardTheme) { ...12 lines };
    public static String retrieveDefinitionFromDatabase(String word,String cardTheme) {
}
}
```

1.1.getConnection() :

```
public static Connection getConnection() {
    try {
        Class.forName( className:"com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection( url:URL, user:USERNAME, password:
    } catch (SQLException | ClassNotFoundException ex) {
        return null;
    }
}
```

Il s'agit d'une méthode Java nommée "getConnection" qui retourne une connexion à une base de données MySQL.

Elle utilise la méthode Class.forName pour charger le pilote "com.mysql.cj.jdbc.Driver", et la méthode DriverManager.getConnection pour établir une connexion à la base de données avec l'URL, USERNAME et PASSWORD spécifiés.

En cas d'exception SQL ou d'exception ClassNotFoundException, la méthode retourne null.

1.2.retrieveWordFromDatabase(String cardTheme):

```
public static String retrieveWordFromDatabase(String cardTheme) {  
    String message = "" ;  
    String sql="select word from "+cardTheme+";";  
    try (Connection conn = Database.getConnection();  
        Statement stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery( string:sql)) {  
        while (rs.next()) {  
            message = rs.getString( string:"word");  
        }  
    } catch (SQLException e) {}  
    return message;  
}
```

Il s'agit d'une méthode Java nommée "retrieveWordFromDatabase" qui récupère un mot à partir d'une base de données.

La méthode prend en entrée une chaîne de caractères "cardTheme" en argument et l'utilise pour créer une requête SQL "select word from " + cardTheme + ";". Elle utilise ensuite la méthode "Database.getConnection()" pour obtenir une connexion à la base de données et créer une déclaration.

La requête est exécutée à l'aide de la méthode executeQuery sur la déclaration et le résultat est stocké dans un objet ResultSet. La méthode parcourt le ResultSet et récupère la valeur de la colonne "word". La valeur est stockée dans la variable "message" et retournée par la méthode.

En cas d'exception SQL, le bloc catch est vide et n'effectue pas de traitement de l'exception.

1.3. retrieveDefinitionFromDatabase(String word, String CardTheme):

```
public static String retrieveDefinitionFromDatabase(String word, String cardTheme) {
    String message = "" ;
    String sql="select definitions from "+cardTheme+" where word='"+word+"'";
    try (Connection conn = Database.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery( string:sql)) {
        while (rs.next()) {
            message = rs.getString( string:"definitions");
        }
    } catch (SQLException e) {}
    return message;
}
```

Cette fonction s'appelle "retrieveDefinitionFromDatabase" et prend en entrée un mot et un thème de carte. Elle retourne une définition sous forme de chaîne de caractères.

La fonction exécute une requête SQL pour sélectionner la définition correspondant au mot et au thème de carte spécifiés dans la base de données. Si la requête réussit, la définition est retournée en tant que message. Sinon, une exception est gérée et aucun message n'est retourné.

IV. Controller

1. Vue general :

```
import javax.swing.*;
import java.awt.*;
public class Controller {
    InitialScreen theView;
    Settings theModel;
    public Controller(InitialScreen theView, Settings theModel) {...4}
    public Controller(InitialScreen theView) {...3 lines }
    public void controllerStartInitialScreen() {...4 lines }
    public void controllerStartGame() {...15 lines }
}
```

2.

Ceci est une classe appelée "Controller" en Java. Elle gère la vue (InitialScreen) et le modèle (Settings) pour un jeu. Il y a deux constructeurs disponibles, un qui prend en charge la vue et le modèle et l'autre qui prend en charge uniquement la vue. La méthode "controllerStartInitialScreen" démarre l'écran initial en utilisant la méthode "initLaunchScreen" de la vue. La méthode "controllerStartGame" démarre le jeu en utilisant la classe "PlayGameWithSingle". Si le nombre de joueurs est égal à un, une fenêtre est créée avec les paramètres de jeu pour démarrer le jeu. Les caractéristiques de la fenêtre telles que la taille, la couleur de fond et la fermeture par défaut sont définies également.

V. Class Card :

1. Vue generale du Code:

```
// Define getters and setters to update card information and info button.
@SuppressWarnings("serial")
public class Cards extends JButton {
    private int id;
    private boolean cardMatched = false;
    public void getIcon(int id) {...2 lines }
    public void setCardNo(int id) {...3 lines }
    public int getCardNo() {...3 lines }
    public void setCardMatchedInfo(boolean cardMatched) {...3 lines }
    public boolean getCardMatchedInfo() {
        return this.cardMatched;
    }
    public void setImageVisible(String no, String themeFolder ) {...6 lines }
    public void changeButtonParams(int[] scores, int time, List<String> names) {...9 li
    public void changeParameters(int score, int time, String name) {...3 lines }
    public void changeParametersMultiPlayer(int score, int score2, String name, String
}
```

Cette classe définit les cartes pour le jeu de mémoire. Elle définit également les getters et les setters pour mettre à jour les informations sur la carte et le bouton d'information. La méthode `setImageVisible` définit l'icône de la carte en utilisant l'image associée à son numéro et au thème choisi. La méthode `changeButtonParams` change les paramètres du bouton d'information en fonction du nombre de joueurs et de leurs scores et temps restants. Les méthodes `changeParameters` et `changeParametersMultiPlayer` définissent les textes affichés sur le bouton d'information pour les modes un joueur ou multi-joueurs respectivement.

2. Getters and Setters:

```
public void getIcon(int id) {  
}  
public void setCardNo(int id) {  
    this.id = id;  
}  
public int getCardNo() {  
    return this.id;  
}  
public void setCardMatchedInfo(boolean cardMatched) {  
    this.cardMatched = cardMatched;  
}  
public boolean getCardMatchedInfo() {  
    return this.cardMatched;  
}
```

- id : un entier représentant le numéro de la carte
- getCardNo() : un getter pour récupérer la valeur de id.
- setCardNo(int id) : un setter pour définir la valeur de id.
- cardMatched : un booléen indiquant si la carte est correspondante ou non.
- getCardMatchedInfo() : un getter pour récupérer la valeur de cardMatched.
- setCardMatchedInfo(boolean cardMatched) : un setter pour définir la valeur de cardMatched.

3. D'autre Methode:

```
public void setImageVisible(String no, String themeFolder ) {
    // en utilisons theme et numero de la card on obtient les images correspondant
    String filename = "Photos/"+themeFolder + "/" + no + ".png";
    File file = new File( pathname:getClass().getClassLoader().getResource( name:filename).getFile());
    setIcon(new ImageIcon( filename:file.getPath()));
}
public void changeButtonParams(int[] scores, int time, List<String> names) {
    // if it is single player
    if (names.size() == 1) {
        changeParameters(scores[0], time, name:names.get( index:0));
    // if it is multiple player.
    } else {
        changeParametersMultiPlayer(scores[0], scores[1], name:names.get( index:0), name2:names.get( index:1));
    }
}
public void changeParameters(int score, int time, String name) {
    setText("Points achieved by " + name + " : " + score + " | Time left : " + time);
}
public void changeParametersMultiPlayer(int score, int score2, String name, String name2) {
    setText("Points " + name + " : " + score + " | " + name2 + " : " + score2);
}
```

Il s'agit du code Java pour une classe qui définit l'image d'un bouton en fonction du thème et du numéro de carte spécifiés, et change le texte du bouton pour afficher des informations différentes en fonction de s'il s'agit d'un jeu en solo ou en multijoueur.

La méthode setImageVisible définit l'image du bouton en construisant d'abord le nom de fichier en fonction du thème et du numéro de carte, puis en créant un objet File avec le chemin de fichier, et enfin en définissant l'icône du bouton sur une nouvelle ImageIcon avec le chemin de fichier.

La méthode changeButtonParams change le texte du bouton en fonction des scores, du temps et des noms des joueurs. S'il n'y a qu'un joueur, le texte affiche le score et le temps restant du joueur. S'il y a plusieurs joueurs, le texte affiche les scores des deux joueurs. Le texte est défini à l'aide de la méthode setText du bouton.

VI. Class PlayGameWithSingle :

1. Vue General du Code

```
import ...9 lines
public class PlayGameWithSingle extends JFrame {
    // initialize class variables.
    private int col;
    private int row;
    private int diffLevel;
    private List<String> playersName;
    private Cards choseMyCard;
    private Cards card1 = null;
    private Cards card2 = null;
    private Cards infoCard;
    private int[] gameScores;
    private int attackCardID;
    private int timeScore;
    private Color backColor;
    private int timeScoreSelected;
    private boolean timeScoreStatus;
    private Timer cardTimeControl;
    private Timer gameTimeControl;
    private List<Cards> Cards;
    private String cardTheme;
    private boolean alreadyExecuted = false;
    private File file_default = new File( pathname: PlayGameWithSingle.class.getResource
    private String DEFAULT_IMAGE = file_default.getPath();
    public PlayGameWithSingle(Settings gameParams) {...50 lines }
    // This list initializes card list. Add action listeners to each card.
    // and return all card as list of objects.
    private List<Cards> initCards(int row, int col, Settings gameParams) {...43 line:

    // this is a method to flip cards back after both are opened.
    private void cardTimeCounter() {...10 lines }
    // This method fills card1 and card2 objects according to actions in the board.
    // Each button has action listeners so that choseMyCard object is filled.
    private void flipCardsSingle() {...32 lines }
    // This method checks both selected cards. They can stay opened or they come back (cl
    private void controlCards() {...64 lines }
    // This method affects game according to difficulty level.
    // Easy : Time Counter increases 1 second.
    // Medium : No effect.
    // Difficult : Time Counter decreases 1 second and an attack Card inserted.
    // If an Attack Card is selected, all opened cards flipped back
    private void startGameEffect(boolean cardMatching) {...25 lines }
    private void startShuffleEffect(Settings gameParams) {...10 lines }
    // This method checks winning state.
    private boolean checkWinning() {...18 lines }
    // This method counts time and change infoButton status (score-time info.)
    private void trackTime() {...34 lines }
    // stop timer.
    private void stopTime() {...4 lines }
```

2. playGameWithSingle:

```
public PlayGameWithSingle(Settings gameParams) {
    // retrieve game information to initialize board and game settings.
    this.col = gameParams.getcolId();
    this.row = gameParams.getrowId();
    this.diffLevel = gameParams.getDiffLevel();
    this.timeScore = gameParams.getTimeInfo();
    this.playersName = gameParams.getPlayersName();
    this.gameScores = new int[gameParams.getPlayersName().size()];
    this.timeScoreStatus = gameParams.getTimeInfo() != 0;
    this.diffLevel = gameParams.getDiffLevel(); // 500:Diff - 1000:Medium -
    this.timeScoreSelected = gameParams.getTimeInfo();
    this.cardTheme = gameParams.getCardTheme();
    this.backColor = gameParams.getBackColor();
    // initialize card Number and List of Card Objects.
    this.Cards = initCards(row, col, gameParams);
    // Create a BigRootPane with border layout.
    // We will put cards and info button inside.
    Container big = getRootPane();
    big.setLayout(new BorderLayout());
    //big.setBackground(Color.white);
    //Set up the board itself
    Container pane = getContentPane();
    pane.setLayout(new GridLayout(rows:row, cols:col));
    pane.setBackground(c:backColor);
    // Create another card object for info button.
    // This will show score and time information.
    Cards infoButton = new Cards();
    // Merge to structure into Root Pane.
    big.add(pane, BorderLayout.CENTER);
}
```

Il initialise les paramètres de jeu à partir des informations fournies dans l'objet "Settings". Il définit la grille de jeu en utilisant un nombre de colonnes et de lignes spécifiées. Il définit également le niveau de difficulté et le temps de score. Il initialise les cartes en appelant une méthode "initCards" qui crée une liste d'objets "Cards". Il crée une structure graphique pour afficher les cartes en utilisant une disposition de grille. Il ajoute également un bouton d'information pour afficher les scores et les informations de temps. Les images par défaut des cartes seront ajoutées à la grille.

2.

```
private List<Cards> initCards(int row, int col, Settings gameParams) {
    // create list of Card objects.
    List<Cards> listOfCards = new ArrayList<>();
    // create list of Card values.
    List<Integer> valuesOfCards = new ArrayList<>();
    // Calculate total number of buttons for the game.
    int pairs = (row * col) / 2;
    //String pairsS = String.valueOf(pairs);
    //System.out.println("Pair is : " + pairsS);
    // Create two card lists as many as pairs.
    for (int j = 1; j <= pairs; j++) {
        valuesOfCards.add(e: j);
        valuesOfCards.add(e: j);
    }

    // Mix card values randomly.
    Collections.shuffle(list: valuesOfCards);
    // This is the ID of the attack card in difficult game.
    attackCardID = valuesOfCards.get(index: 0);
    System.out.println("Attacked Card : " + attackCardID);
    // add actions to each card. Then, return card objects.
    for (Integer valuesOfCard : valuesOfCards) {
        // Create Card object for each image.
        Cards mySelect = new Cards();
        mySelect.setCardNo(id: valuesOfCard);
        // Add action listener to change image of the card.
        mySelect.addActionListener((ActionEvent ae) -> {
            // assign mySelect as chosen card and then call flipCards method.

            choseMyCard = mySelect;
            // Flip cards will be called after any of cards is pressed.
            flipCardsSingle();
        });
    }
    // Add all buttons into one list to process them in a board.
    listOfCards.add(e: mySelect);

    //System.out.println(listOfCards.toString());
    return listOfCards;
}
```

Cette méthode initialise la liste de cartes. Elle ajoute des écouteurs d'action à chaque carte et retourne toutes les cartes sous forme de liste d'objets. La méthode crée d'abord une liste d'objets Cartes, puis une liste de valeurs de cartes. Le nombre total de boutons pour le jeu est calculé en divisant le nombre de lignes par deux. Ensuite, deux listes de cartes sont créées, chacune

aussi grande que le nombre de paires. Les valeurs de cartes sont mélangées de manière aléatoire. L'ID de la carte d'attaque est défini pour le niveau de difficulté difficile. Des actions sont ajoutées à chaque carte et toutes les cartes sont ajoutées à une liste pour être utilisées dans le tableau. La méthode retourne la liste des objets Cartes.

3. CardTimeCounter():

```
// this is a method to flip cards back after both are opened.
private void cardTimeCounter() {

    //set up the timer
    cardTimeControl = new Timer( delay:750, (ActionEvent ae) -> {
        controlCards();
    });

    cardTimeControl.setRepeats( flag:false);
    cardTimeControl.start();
}
```

cardTimeCounter method:

- Cette méthode configure un minuteur qui s'exécute une seule fois après 750 millisecondes.
- Il utilise la classe Timer et passe une instance de l'interface ActionListener en tant qu'argument.
- Lorsque le minuteur atteint le délai de 750 millisecondes, il invoque la méthode controlCards.
- La propriété repeats du minuteur est définie sur false, ce qui signifie qu'il ne se répète pas.

Enfin, la méthode start du minuteur est appelée pour démarrer le compteur.

4. flipCardsSingle():

```
// Each button has action listeners so that choseMyCard object is filled
private void flipCardsSingle() {
    // This condition helps us to understand first card is selected.
    if (card1 == null && card2 == null) {
        // start timer of the game once.
        if (!alreadyExecuted) {
            trackTime();
            alreadyExecuted = true;
        }
        System.out.println(x: "Card 1 is selected");
        // After card is chosen, image need to be showed.
        card1 = choseMyCard;
        // In this step, image was taken according to card no.
        String no = String.valueOf(i: card1.getCardNo());
        // Then, card-button displays image.
        card1.setImageVisible(no, themeFolder: cardTheme);
    }
    // This case helps us to select second card.
    else if (card1 != null && card1 != choseMyCard && card2 == null) {
        // Assign selected card as my new card.
        card2 = choseMyCard;
        // Then, get card no of it.
        String no = String.valueOf(i: card2.getCardNo());
        // Finally, card object shows its image.
        card2.setImageVisible(no, themeFolder: cardTheme);
        // start time to come back.
        cardTimeCounter();
    }
}
```

flipCardsSingle method:

Cette méthode gère le processus de retournement d'une seule carte.

- Si aucune carte n'a été sélectionnée (card1 est égal à null et card2 est égal à null), la première carte est définie sur la carte choisie et son image est affichée.

- Si la première carte a déjà été sélectionnée (card1 est différent de null et card1 est différent de la carte choisie et card2 est égal à null), la seconde carte est définie sur la carte choisie, son image est affichée et la méthode cardTimeCounter est appelée pour démarrer le minuteur.

5. startGameEffect():

```
private void startGameEffect(boolean cardMatching) {
    if (cardMatching) {
        // increase time information by one.
        if (diffLevel == 500) {
            timeScore++;
        }
    } else {
        // decrease time information by one.
        // and there is a card pair. If one them comes, all opened cards become closed.
        if (diffLevel == 2000) {
            System.out.println("In the difficult level ");
            timeScore--;
            if (card1.getCardNo() == attackCardID | card2.getCardNo() == attackCardID) {
                //System.out.println("Attacked Card : " + attackCardID);
                for (Cards comp : Cards) {
                    comp.setIcon(new ImageIcon(filename: DEFAULT_IMAGE));
                    comp.setEnabled(b: true);
                    comp.setCardMatchedInfo(cardMatched: false);
                }
                // make game score zero.
                gameScores[0] = 0 ;
            }
        }
    }
}
```

startGameEffect method:

Cette méthode gère l'effet du début du jeu en fonction du résultat de la correspondance des cartes.

Si les cartes correspondent (cardMatching est égal à true), le temps augmente de 1 si la difficulté est égale à 500.

Si les cartes ne correspondent pas (cardMatching est égal à false), le temps diminue de 1 si la difficulté est égale à 2000 et une paire de cartes est présente.

Si une des cartes est la carte d'attaque (définie par attackCardID), toutes les cartes ouvertes deviennent fermées et le score de jeu est réinitialisé à zéro.

6. controlCards():

```
// This method checks both selected cards. They can stay opened or they come back (closed).
private void controlCards() {
    if (card1.getCardNo() == card2.getCardNo()) {
        int no = card1.getCardNo();
        String word = Database.retrieveWordFromDatabase( no ,cardTheme);
        String definition = Database.retrieveDefinitionFromDatabase(word,cardTheme);
        JOptionPane optionPane = new JOptionPane("Word: " + word + " \t\nDefinition: " + definition, messageType:JOptionPane.QUESTION_MESSAGE);
        JDialog dialog = new JDialog();
        dialog.setAlwaysOnTop( alwaysOnTop: true);
        dialog.setModal( modal: true);
        dialog.setContentPane( contentPane: optionPane);
        dialog.setDefaultCloseOperation( operation: JDialog.DISPOSE_ON_CLOSE);
        dialog.setTitle( title: "Word and Definitions");
        dialog.pack();
        dialog.setLocationRelativeTo( c: null);
        dialog.setVisible( b: true);
        JButton closeButton = new JButton( text: "Close");
        JButton okButton = new JButton( text: "OK");
        okButton.addActionListener((ActionEvent e) -> {
            dialog.dispose();
        });
        optionPane.setOptions(new Object[] {okButton});
        Timer timer = new Timer( delay: 5000, (ActionEvent e) -> {
            dialog.dispose();
        });
        optionPane.add( comp: closeButton);
        timer.setRepeats( flag: false);
        timer.start();
    }
}
```

La méthode controlCards est utilisée pour contrôler le comportement des cartes. Si les numéros de cartes card 1 et card2 sont les mêmes, alors une fenêtre JOptionPane est affichée avec le mot et sa définition correspondants récupérés à partir de la base de données. Un bouton "Fermer" et "OK" est affiché avec la fenêtre et un timer de 5 secondes est défini pour fermer la fenêtre. Si les cartes sont associées, elles sont désactivées et le score du jeu est mis à jour. Si le jeu est gagné, une boîte de dialogue est affichée pour demander si le joueur souhaite rejouer ou quitter. Dans le cas contraire, les images des deux cartes sont remises à leur état par défaut. Enfin, les variables card1 et card2 sont réinitialisées à null.

7. tracktime():

```
// this method counts time and change infoButton status (score-time info.)
private void trackTime() { // Define timer with 1 second period.
    gameTimeControl = new Timer(delay:1000, new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            //System.out.println("test timer has started");
            // Control timeCounter is true or false. If it is true, time will be decreased.
            // Otherwise, only infoButton will be updated.
            if (timeScoreStatus) {
                timeScore--;
                //infoCard.changeParameters(score, remTime, playerNameOne);
                infoCard.changeButtonParams(scores:gameScores, time:timeScore, names:playersName);
                // check time information.
                if (timeScore <= 0) {
                    stopTime();
                    int response = JOptionPane.showConfirmDialog(parentComponent:null, message:"Time is UP. ",
                        title:"Do you want to play this game again ? ", optionType:JOptionPane.YES_NO_CANCEL_OI
                    if (response == 0) {
                        // start new game
                        dispose();
                        InitialScreen newGame = new InitialScreen();
                        newGame.initLaunchScreen();
                    } else if (response == 1) {
                        // exit from the game.
                        System.exit(status:0);
                    }
                }
            } else { //infoCard.changeParameters(score, remTime, playerNameOne);
                infoCard.changeButtonParams(scores:gameScores, time:timeScore, names:playersName);
            }
        }
    });
    gameTimeControl.start();
}
```

Cette méthode appelée "trackTime" définit un minuteur d'une période de 1 seconde. L'événement `ActionListener` déclenche l'action de décrémenter le temps restant si l'indicateur "timeScoreStatus" est vrai. Si le temps atteint zéro, un message d'invitation à rejouer le jeu apparaît et si l'utilisateur choisit de quitter le jeu, celui-ci se ferme. La méthode "infoCard.changeButtonParams" est appelée à chaque tick du minuteur pour mettre à jour les scores et le temps restant.

VII. ScoreUpdate

classe appelée "ScoreUpdate" qui est utilisée pour mettre à jour les scores des joueurs dans une base de données. La classe contient plusieurs méthodes pour réaliser cette tâche.

1. Constructeur ScoreUpdate:

Le constructeur de la classe ScoreUpdate prend en entrée cinq paramètres : le temps restant, le nombre de lignes, le nombre de colonnes, le temps sélectionné et le nom du joueur. Ces paramètres sont stockés dans des variables de la classe pour une utilisation ultérieure.

```
public ScoreUpdate(int remTime, int row, int col, int selectedTime, String playerName) {  
  
    this.remTime = remTime;  
    this.row = row;  
    this.col = col;  
    this.selectedTime = selectedTime;  
    this.playerName = playerName;  
}
```

2. Méthode checkAndUpdate():

La méthode checkAndUpdate() est utilisée pour vérifier et mettre à jour le score du joueur. Elle appelle la méthode scoreCalculation() pour calculer le score final du joueur, puis appelle la méthode scoreControl() pour confirmer que le nouveau score doit être inséré dans la table de scores. Si le score est mis à jour avec succès, la méthode affiche un message de félicitations.

```
public void checkAndUpdate () {  
  
    double finalScore = scoreCalculation(remTime, row, col, selectedTime);  
    // check and update score  
    boolean result = scoreControl( score:finalScore, playerName);  
    if (result) {  
        System.out.println( x:"Score will UPDATED...!! CONGRATULATIONS...");  
    }  
}
```

3. Méthode scoreCalculation():

La méthode scoreCalculation() est utilisée pour calculer le score final du joueur en fonction du temps restant, du nombre de lignes et de colonnes, et du temps sélectionné. Elle prend en entrée ces paramètres et calcule le temps utilisé en soustrayant le temps restant du temps sélectionné. Le score final est calculé en multipliant la taille du plateau de jeu (nombre de lignes fois nombre de colonnes) par le carré de la taille du plateau divisé par le temps utilisé. Si le temps sélectionné est égal à zéro, le score final est défini à zéro.

```

// this method calculates score of the game.
private double scoreCalculation(int remTime, int row, int col, int selectedTime) {
    int size = col * row;
    double finalScore;
    System.out.println("Remained time is " + remTime);
    System.out.println("Size is " + size);
    System.out.println("Total time is " + selectedTime);
    // Score Calculation
    int usedTime = selectedTime - remTime;
    System.out.println("usedTime");
    if (selectedTime == 0) {
        finalScore = 0;
    } else {
        finalScore = size * ((double) size / usedTime);
        System.out.println("Final Score" + finalScore);
    }
    return finalScore;
}

```

4. Méthode scoreControl():

La méthode scoreControl() est utilisée pour insérer le nouveau score dans la table de scores de la base de données. Elle prend en entrée le score final et le nom du joueur et crée une instruction SQL pour insérer ces valeurs dans la table. Elle utilise ensuite une connexion à la base de données et un objet PreparedStatement pour exécuter l'instruction SQL. Si l'insertion est réussie, la méthode renvoie true, sinon elle renvoie false.

```
// this method confirms new score is to be inserted leaderboard table.
private boolean scoreControl(double score, String playerName) {
    System.out.println("Final score is " + score);
    // get current date
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-YYYY");
    LocalDateTime now = LocalDateTime.now();
    String game_date = dtf.format(now);
    System.out.println("game_date");
    // create SQL statement
    String sql = "INSERT INTO accounts (username, score, game_date) VALUES (?, ?, ?)";
    try (Connection conn = Database.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
        // set parameters
        pstmt.setString(1, playerName);
        pstmt.setDouble(2, score);
        pstmt.setString(3, game_date);
        // execute SQL statement
        int rowsUpdated = pstmt.executeUpdate();
        // check if score was successfully inserted
        return rowsUpdated > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

VIII. CustomTableCellRenderer

La classe CustomTableCellRenderer étend la classe DefaultTableCellRenderer et implémente la méthode getTableCellRendererComponent. Cette classe est utilisée pour personnaliser l'affichage des cellules dans une JTable en alternant les couleurs de fond des lignes.

La méthode getTableCellRendererComponent prend en paramètres la table, la valeur de la cellule, l'état de sélection, l'état de mise au point, la ligne et la colonne de la cellule.

1. getTableCellRendererComponent

de la classe parent pour initialiser le composant par défaut. Ensuite, elle détermine la couleur de fond en fonction du numéro de ligne. Si le numéro de ligne est pair, la couleur de fond est COLOR1, sinon elle est COLOR2. Enfin, la méthode renvoie le composant mis à jour avec la couleur de fond correcte.

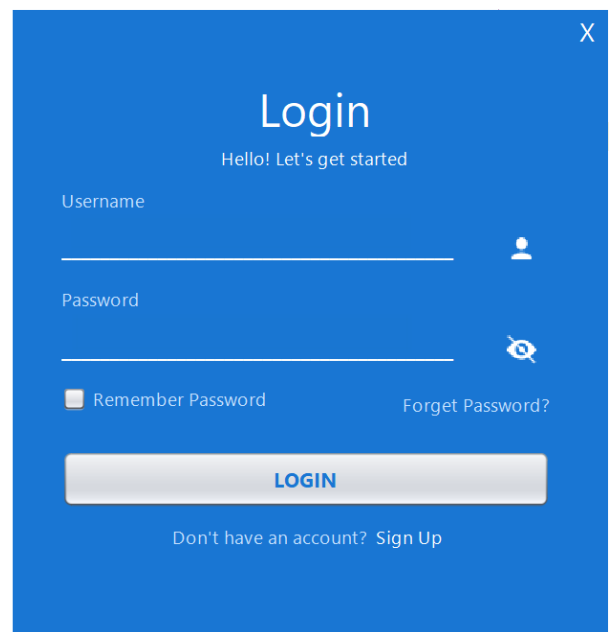
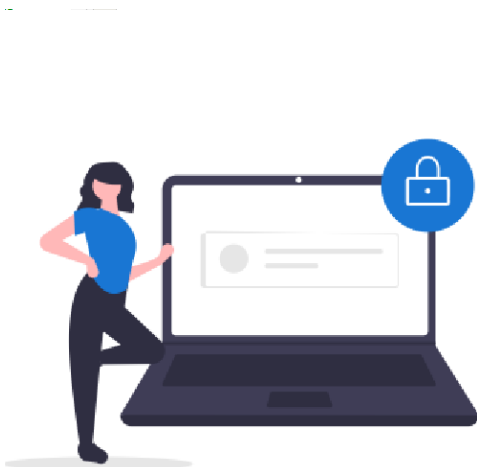
2. CustomTableCellRenderer

est utilisée pour personnaliser l'affichage des cellules dans une JTable en alternant les couleurs de fond des lignes. Cette classe est utilisée en tant que rendu de la table, ce qui signifie qu'elle est appelée pour chaque cellule de la table.

```
class CustomTableCellRenderer extends DefaultTableCellRenderer {  
    private static final Color COLOR1 = new Color( r:245, g:245, b:245); // light gray  
    private static final Color COLOR2 = new Color( r:255, g:255, b:255); // white  
    private int row;  
  
    @Override  
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,  
        boolean hasFocus, int row, int column) {  
        Component c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column)  
        this.row = row;  
        c.setBackground(row % 2 == 0 ? COLOR1 : COLOR2);  
        return c;  
    }  
}
```

Lien Github : <https://github.com/useAllasS/MiniProjet-Matching-Game>

GAME CAPTURES:



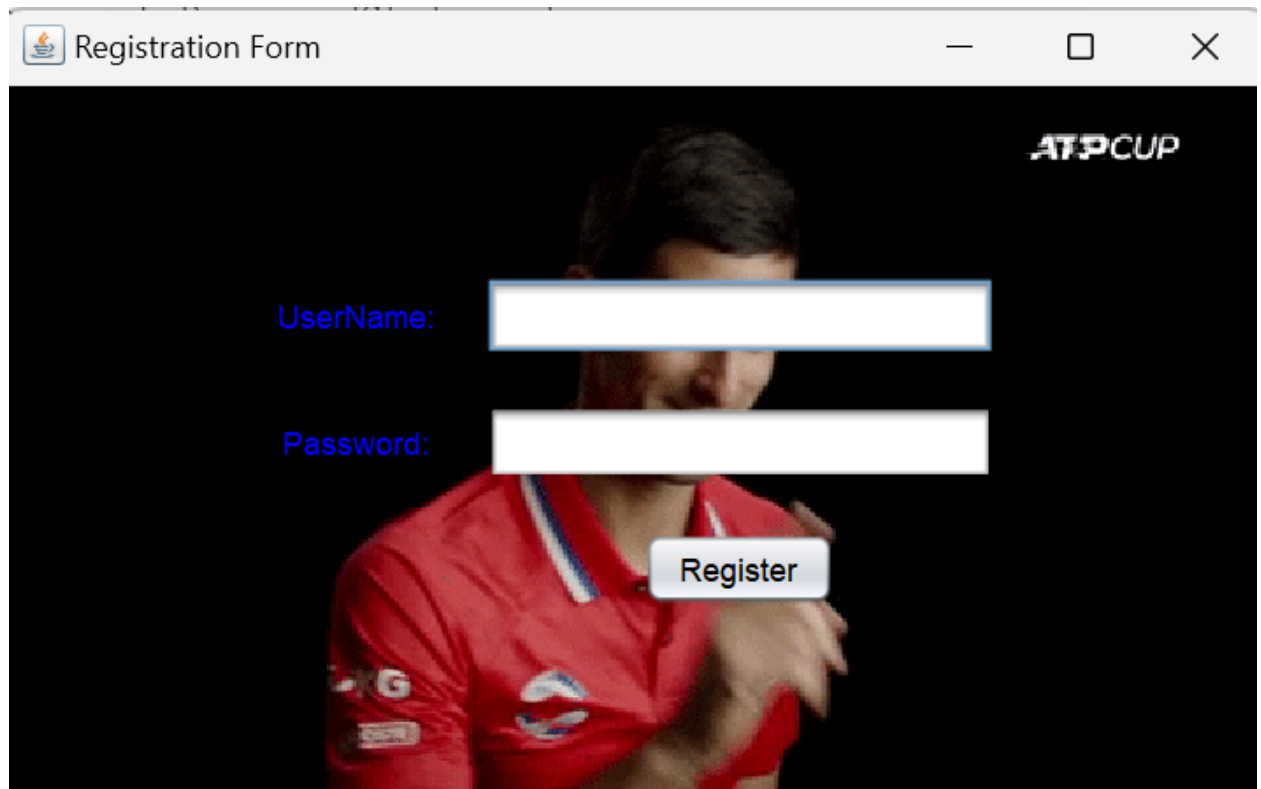
Registration Form

ATP CUP

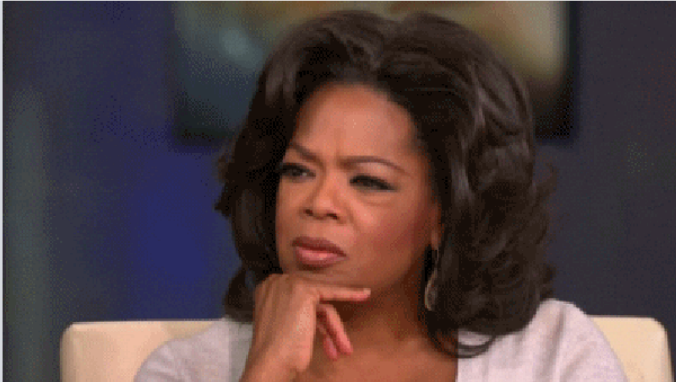
UserName:

Password:

Register

A screenshot of a web browser window titled "Registration Form". The background is a dark image of a tennis player in a red shirt. In the top right corner, the "ATP CUP" logo is visible. The form contains two labels, "UserName:" and "Password:", both in blue text. Each label is followed by a white input field. Below the input fields is a white button with the text "Register".

Memory Matching Game



Rules About Memory Game

Difficulty Level: ☐ Easy ☐ Medium ☐ Difficult

Time Setting: ☐ 0 sec ☐ 10 sec ☐ 60 sec ☐ 90 sec ☐ 120 sec

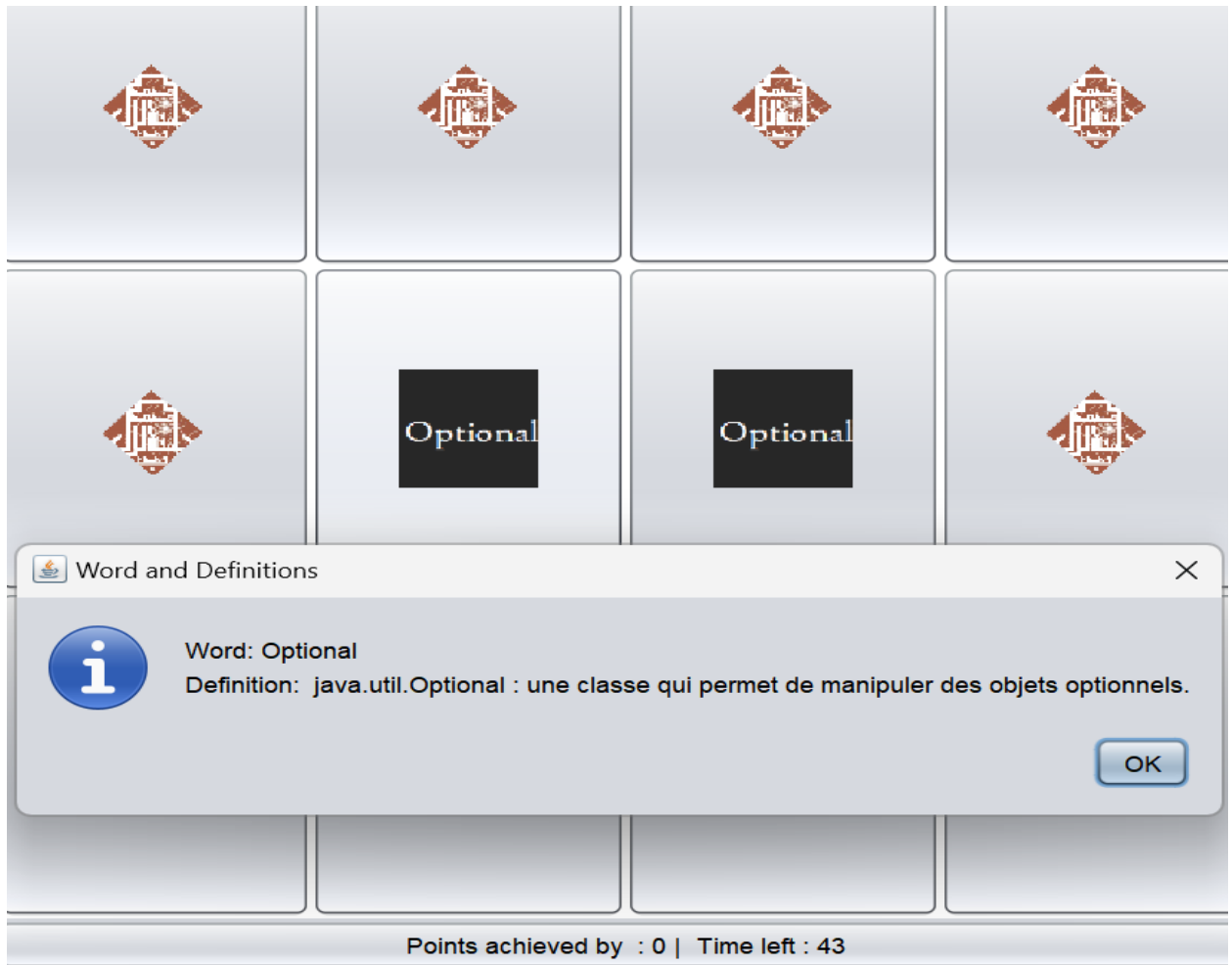
Opponent Type: ☐ Double

of rows # of cols

Card Theme Change Background High Scores Start Exit

Leaderboard

Username	Total Points	Date
NITA1	9.0	2022-02-14
sawsan	8.0	2022-02-12
nita	8.0	2022-02-15
fouad	7.0	2022-02-13
salam	6.0	2022-02-11



XI. Ideas

L'idée d'amélioration du jeu peut être réalisée par plusieurs façons, quelques exemples:

1. La généralisation du thème des cartes pour accepter plus de thèmes, d'autres concepts comme la conception orientée objet, algorithmique, probabilité ...
2. La production d'autres façons de jouer, ce qui signifie une autre méthode de jeu. En prenant par exemple le jeu de mémoire ou le matching game basé sur la mémorisation de l'emplacement des cartes et puis effectuer le match et ainsi de suite, si on prend le principe de ce jeu et introduit le Guess Game ou en mémorisant l'emplacement des mots d'un texte ou marcher selon la compatibilité des objets.

3. Développer l'application pour accepter le jeu avec d'autres personnes online c-a-d connecte au jeu en même temps.
4. On marque aussi le manque de motivation comme cadeaux