

Informe de Laboratorio Final

Tema: Proyecto Final - Videojuego

Nota

Estudiante	Escuela	Asignatura
Eduardo Portugal & Hernan Choquehuanca eportugalpor@unsa.edu.pe hchoquehuancaz@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
Final	Proyecto Final - Videojuego	72 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 22 Enero 2024	Al 29 Enero 2024

1. Tarea

- Terminar trabajo final - VideoJuego

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 22H2 64 bits.
- Visual Studio Code 1.82.2.0.
- JDK 17 Full 64-Bits 17.0.7.7.
- Git 2.41.0.2.
- Cuenta en GitHub con el correo institucional.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/hernanchoquehuanca/fp2-23b.git>
- URL para el laboratorio final en el Repositorio GitHub.
- https://github.com/hernanchoquehuanca/fp2-23b/tree/main/fase03/proyecto_final

4. Laboratorio Final

- A continuación se mostrara el desarrollo del ejercicio de laboratorio:

4.1. Diagrama UML de clases

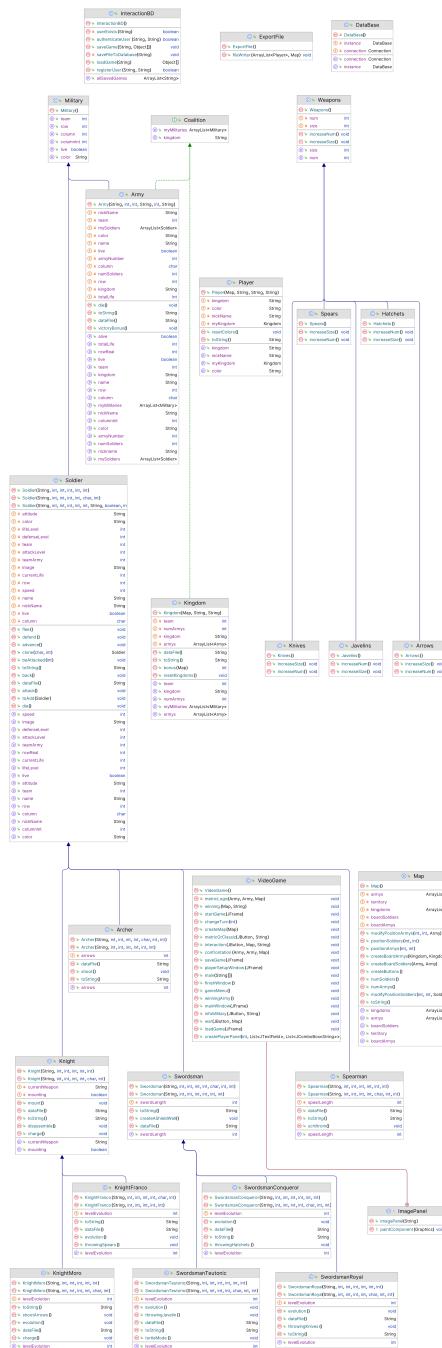


Figura 1: Click aquí para ver Diagrama UML de clases completo - Demasiado Grande

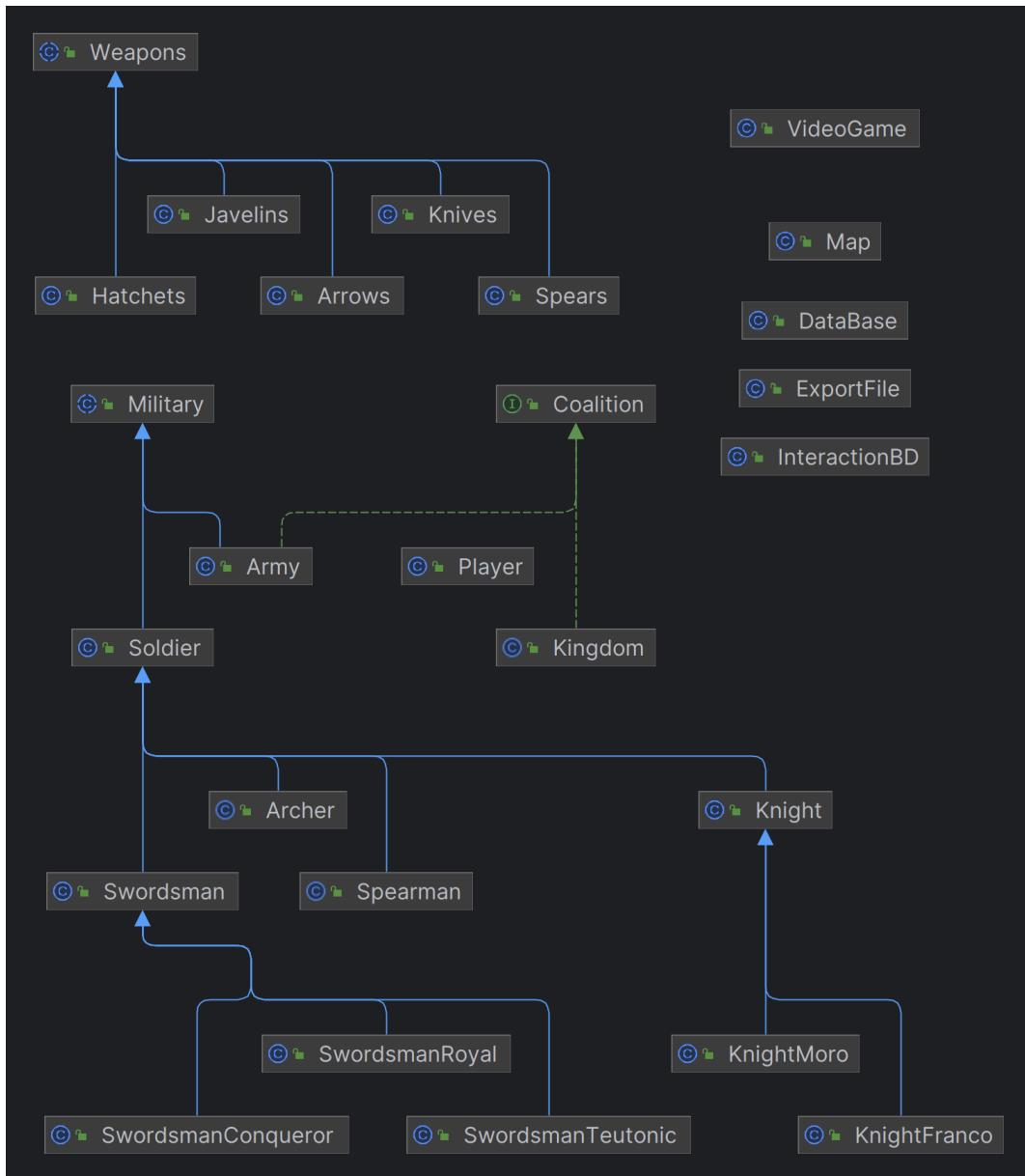


Figura 2: Relaciones de herencia entre las clases

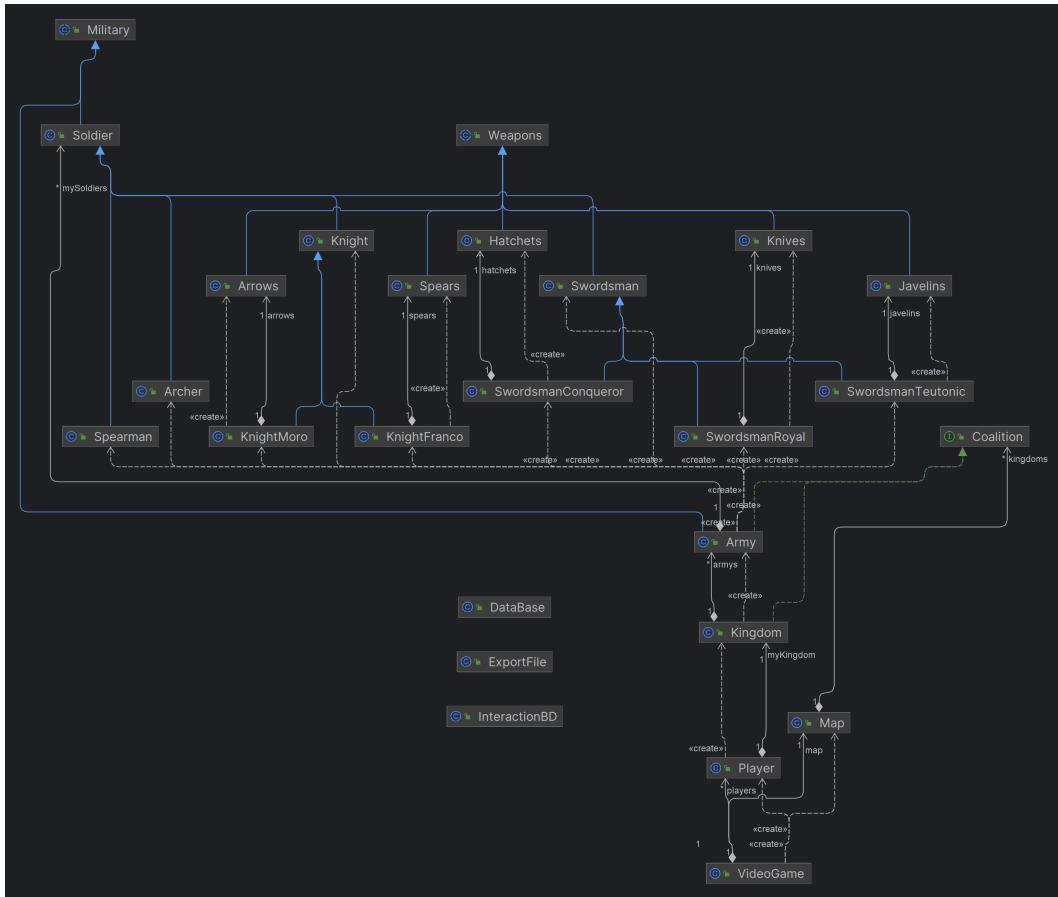


Figura 3: Relaciones de dependencia entre las clases

- Se puede apreciar la clase **Soldier** y todas sus subclases, ademas de las subclases de unidades especiales.
- Todas las clases de armas son subclases de la clase abstracta **Weapons.java**.
- Tambien se aprecia la composicion de la clase **Soldier**, con respecto a la clase **Army**. Asimismo, estas dos clases son subclases de la clase abstracta **Military.java**.
- De igual forma se aprecia la composicion de la clase **Army**, con respecto a la clase **Kingdom**. Asimismo, estas dos clases comparten la interface **Coalition.java**.
- Existe la clase **Map** que representa el tablero del **VideoJuego**.
- Se creo la clase **DataBase.java** con la estructura de diseño de singleton para tener una unica conexion definida hacia la base de datos.
- Tambien existe la clase **InteractionBD.java** a traves de la cual se realizan todas la consultas, imports, entre otros, hacia la base de datos.
- De igual forma, se presenta el archivo **VideoGame.java** que permitira la interaccion de todas las otras clases y el funcionamiento del **VideoJuego**.

4.2. Clases Utilizadas

- Archer.java
- Army.java
- Arrows.java
- Coalition.java
- DataBase.java
- ExportFile.java
- Hatchets.java
- InteractionBD.java
- Javelins.java
- Kingdom.java
- Knight.java
- KnightFranco.java
- KnightMoro.java
- Knives.java
- Map.java
- Military.java
- Player.java
- Soldier.java
- Spearman.java
- Spears.java
- Swordsman.java
- SwordsmanConqueror.java
- SwordsmanRoyal.java
- SwordsmanTeutonic.java
- VideoGame.java
- Weapons.java

4.3. Interfaz Gráfica

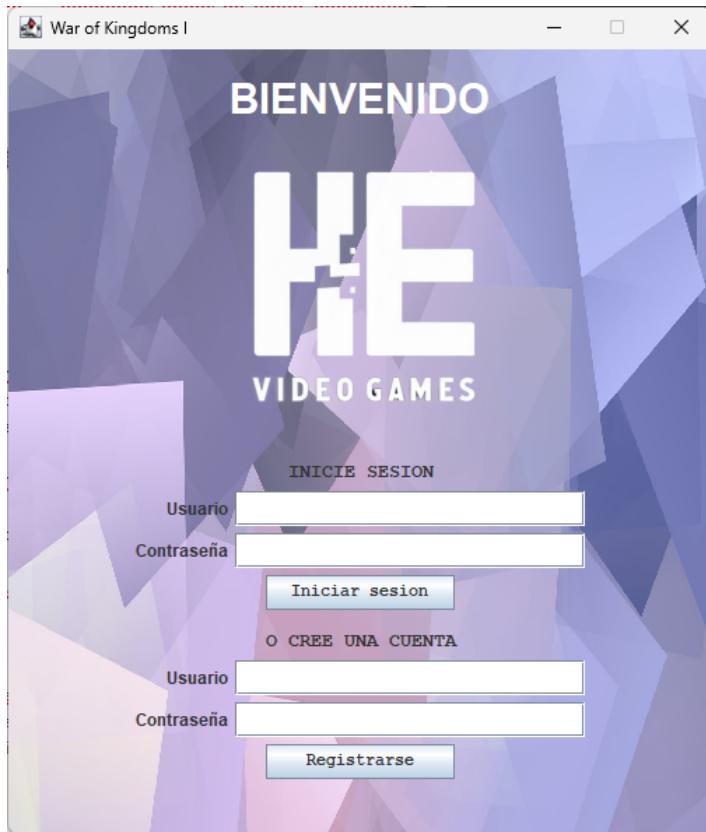


Figura 4: Menú de Sesión del VideoJuego

- Se crea una ventana principal (window0) con un tamaño específico, título, ubicación central, y diseño no redimensionable.
- Se establece una imagen de fondo en la ventana principal.
- Se añade un título "BIENVENIDO" y una imagen de login con un logo escalado al centro de la ventana.
- Se crean botones para iniciar sesión y registrarse, con un diseño específico y estilo de fuente.
- Se definen campos de texto para el nombre de usuario y contraseña, junto con etiquetas descriptivas.
- Se configuran acciones al presionar `.Enter` en los campos de texto para simular un clic en el botón de inicio de sesión.
- Se organizan los elementos en paneles y se agregan a la ventana principal.
- Se manejan eventos de clic en los botones de inicio de sesión y registro, verificando la autenticación y registrando nuevos usuarios mediante interacción con una base de datos (InteractionBD).
- Si la autenticación o el registro son exitosos, se abre una ventana principal (mainWindow).
- La ventana principal se hace visible al final.

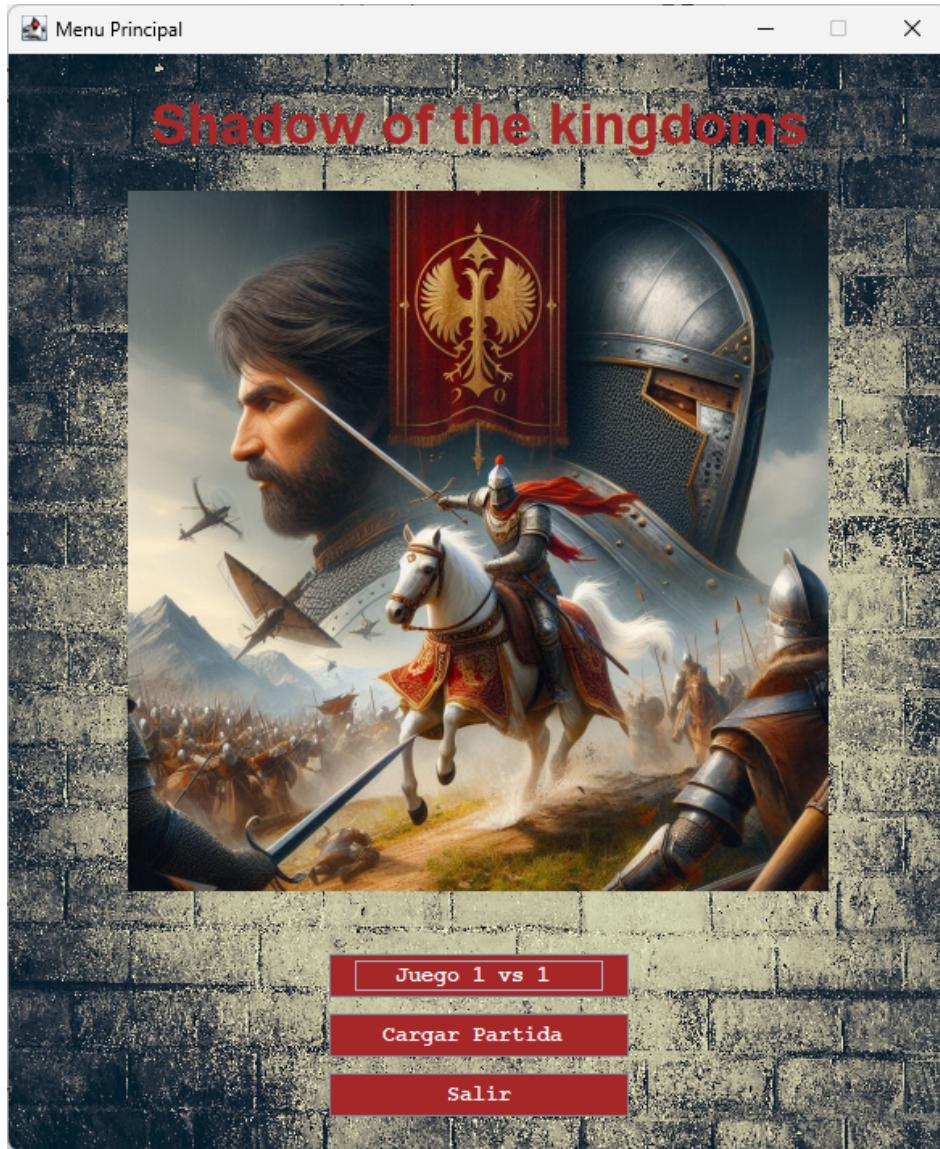


Figura 5: Menú de Principal del VideoJuego

- El código cierra la ventana actual, crea y muestra una nueva ventana para el menú principal de un juego llamado "Shadow of the Kingdoms".
- La ventana contiene un título, una imagen de portada, y botones para opciones como iniciar un juego 1 vs 1, cargar una partida, o salir del juego.
- La interfaz gráfica utiliza colores y fuentes específicos para dar estilo a la presentación.
- El código también define acciones para los botones, como abrir una ventana de configuración de jugador al hacer clic en el primer botón, cargar una nueva partida al hacer clic en el segundo botón, y cerrar la aplicación al hacer clic en el tercer botón.



Figura 6: Menú de Personalización del VideoJuego

- La función `playerSetupWindow` toma una ventana existente (`window`), la cierra y crea una nueva ventana para la configuración de jugadores.
- Se inicializa el mapa y una lista de jugadores.
- Se crean listas para almacenar componentes de la interfaz gráfica, como campos de texto y cuadros de selección.
- Se configura la apariencia básica de la nueva ventana, incluyendo una imagen de fondo.
- Se crea un panel principal vertical que contiene estructuras de ingreso para dos jugadores.
- Cada estructura de ingreso se crea mediante la función `createPlayerPanel`, que incluye un número de jugador, un campo de texto para el nombre, y cuadros de selección para el reino y el color.
- Se añade un botón "Comenzar partida" que recoge la información ingresada y crea objetos de jugador con esos datos. Luego, inicia el juego.
- La función `createPlayerPanel` retorna un panel con los elementos de entrada y selección para un jugador específico.

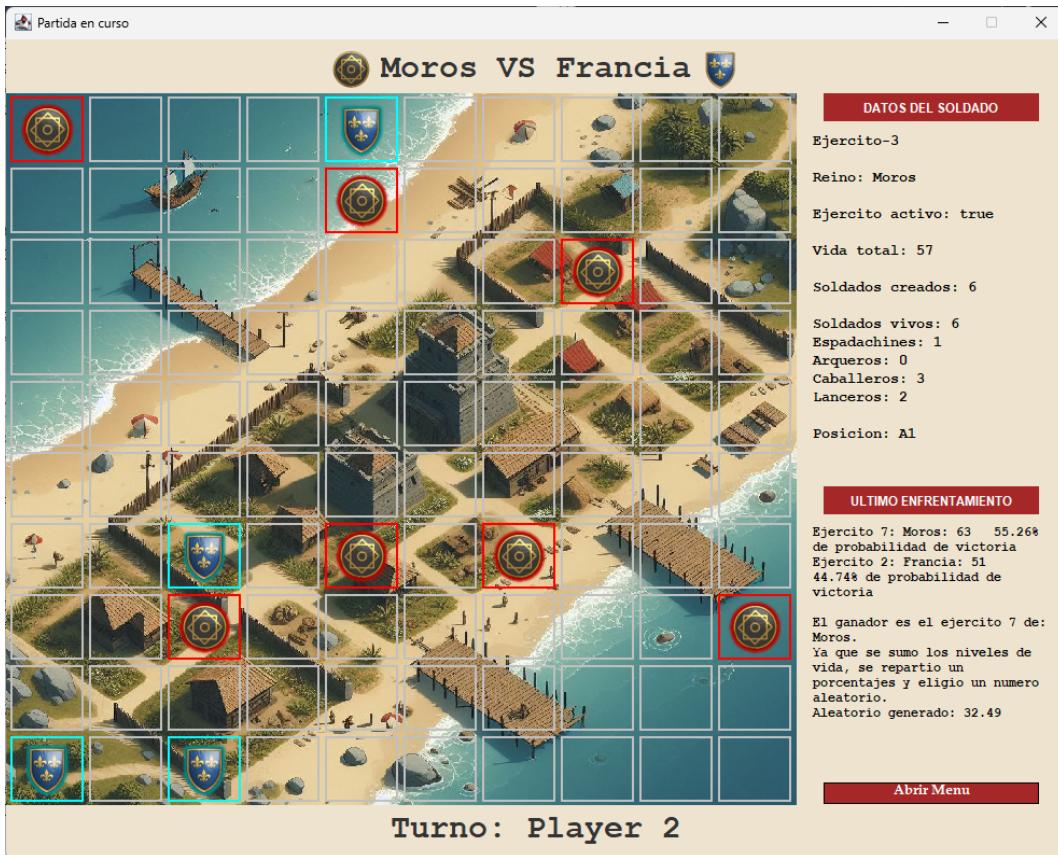


Figura 7: Ventana del VideoJuego - Ejercitos

- La función startGame toma una ventana existente (window), la cierra y crea una nueva ventana para la partida en curso.
- Se configura la apariencia básica de la nueva ventana, incluyendo el tamaño, la ubicación, y la no redimensionabilidad.
- Se crea un encabezado que muestra los reinos en guerra (kingdom1 VS kingdom2) y las imágenes de los reinos.
- Se crea y muestra un mapa utilizando la función createMap y se añade a la ventana.
- Se crea un panel de información que muestra datos sobre los ejércitos y soldados.
- Se añaden paneles y etiquetas para mostrar información relevante, como datos del soldado y el resultado de la batalla.
- Se añade un botón “Abrir Menú” que llama a la función gameMenu al hacer clic.
- Se configura la parte inferior de la ventana para mostrar el turno actual del jugador.
- Se añade un listener para la tecla Escape, que llama a la función gameMenu al presionar Escape.
- La ventana se hace visible al final.
- Al llamar a la función createMap, se crea un panel (mapArmys) con un diseño de cuadrícula para representar el mapa de ejércitos.

- El mapa de ejércitos se inicializa con un nuevo objeto de mapa proporcionado como parámetro.
- Se llama al método createBoardArmys del objeto map para generar la configuración inicial del tablero de ejércitos.
- Este método utiliza información sobre los reinos de los jugadores para establecer la distribución inicial de los ejércitos en el tablero.
- Se configuran botones en el panel de ejércitos (mapArmys). Cada botón se asigna a un Action-Listener llamando al método interaction al hacer clic, y se añade un MouseListener para llamar al método infoMilitary cuando el mouse entra en el área del botón.
- Los botones se añaden al panel de ejércitos, completando así la representación visual del mapa de ejércitos.

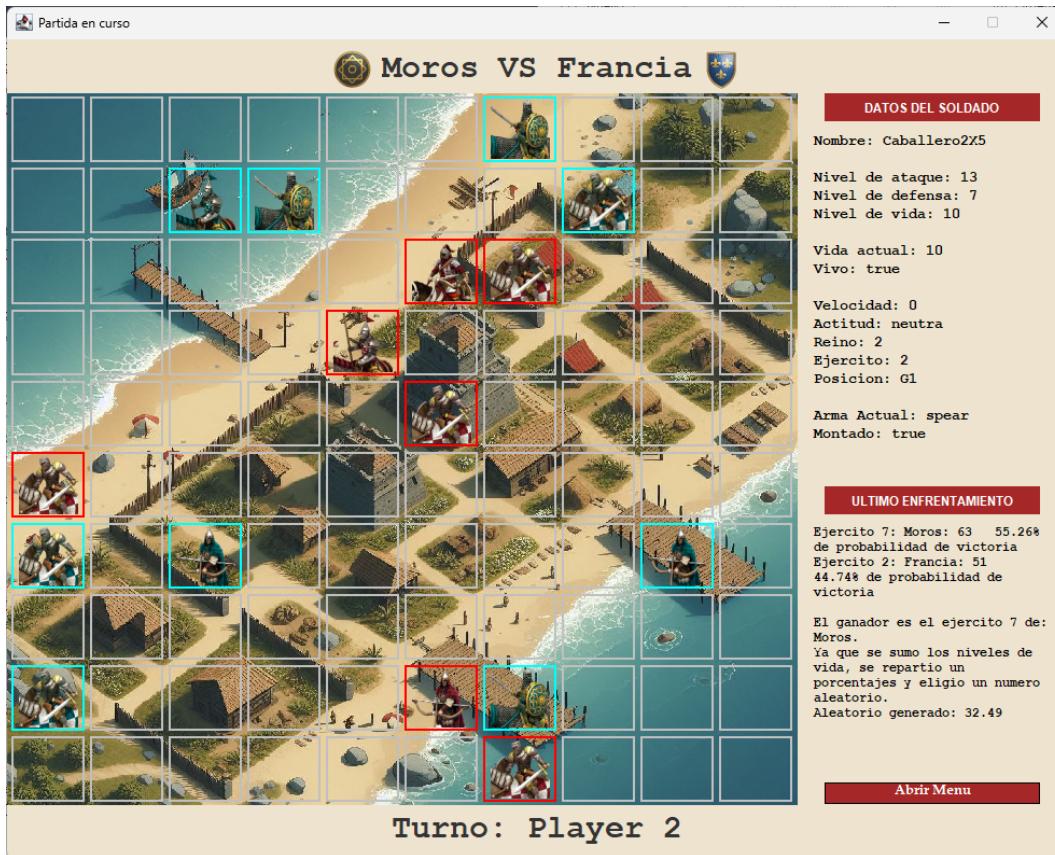


Figura 8: Ventana del VideoJuego - Soldados

- Al enfrentarse 2 ejércitos, se llama a la función confrontation donde se crea un nuevo panel (mapSoldiers) con un diseño de cuadrícula para representar el mapa de soldados.
- El método createBoardSoldiers del objeto map se llama para configurar la distribución inicial de los soldados en el tablero, utilizando información sobre los ejércitos aliados y enemigos.
- Se establece la variable then como nula.
- Se configuran botones en el panel de soldados (mapSoldiers). Cada botón se asigna a un Action-Listener llamando al método interaction al hacer clic, y se añade un MouseListener para llamar al método infoMilitary cuando el mouse entra en el área del botón.
- Los botones se añaden al panel de soldados.
- Se elimina cualquier componente existente en el panel de juego (mapSpace).
- El panel de soldados se añade al panel de juego.
- Se actualiza y repinta la ventana principal (window2) para reflejar los cambios, mostrando el tablero de los soldados.

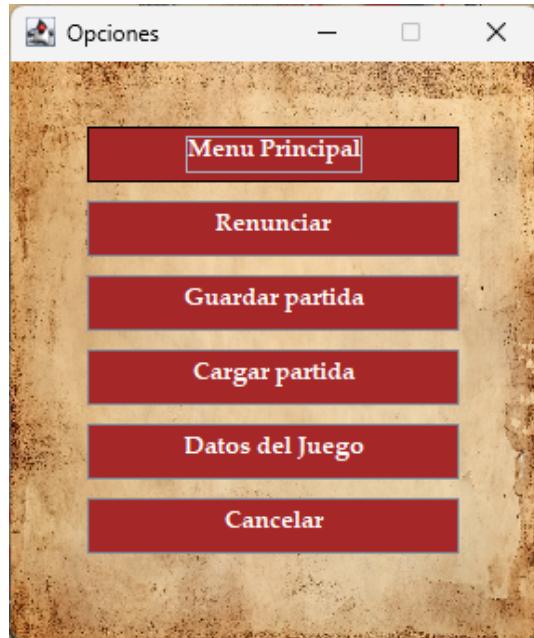


Figura 9: Menú de Pausa

- Se crea una nueva ventana (windowGameMenu) con configuraciones específicas de tamaño, cierre, ubicación y no redimensionamiento.
- Se establece una imagen de fondo en la ventana utilizando un panel ImagePanel.
- Se crea un panel (panel) para organizar los botones en forma de columna utilizando BoxLayout en el eje Y.
- Se crean varios botones para diferentes opciones del menú, cada uno con su acción correspondiente al hacer clic.
- Los botones se añaden al panel, separados por espacios rígidos para dar un formato adecuado.
- Se configuran las propiedades visuales de los botones, como el color del texto, el fondo y la fuente.
- Se añade un borde alrededor del panel para darle un aspecto estéticamente agradable.
- El panel se añade a la ventana en la región central.
- La ventana se hace visible al final.

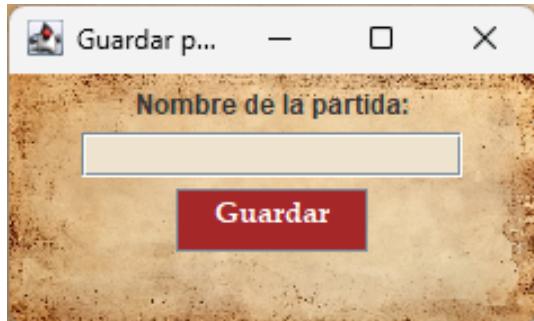


Figura 10: Menú de Guardar la partida

- Se crea una nueva ventana (windowSaveGame) con configuraciones específicas de tamaño, ubicación, cierre y diseño de flujo (FlowLayout).
- Se establece una imagen de fondo en la ventana utilizando un panel ImagePanel.
- Se crean componentes de interfaz de usuario, como una etiqueta (lblNombre) para indicar el propósito del campo de texto, un campo de texto (txtNombre) para ingresar el nombre de la partida, y un botón (btnGuardar) para realizar la acción de guardar.
- Se configuran propiedades visuales, como el fondo y la fuente, para algunos de los componentes.
- Se añade un ActionListener al botón de guardar (btnGuardar).
- Cuando se presiona el botón, se obtiene el texto ingresado en el campo de texto y se utiliza para guardar la partida utilizando el método InteractionBD.saveGame(). Luego, la ventana de guardar se cierra.
- Los componentes se añaden a la ventana en el orden deseado.
- La ventana de guardar se hace visible al final.

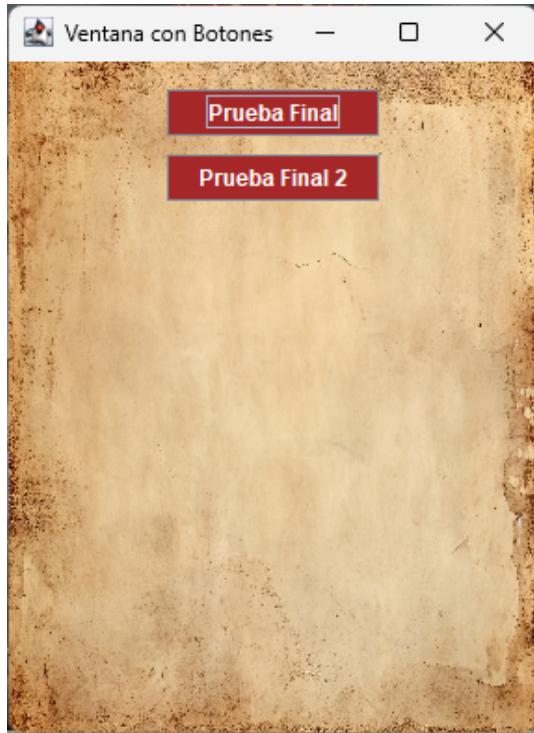


Figura 11: Menú de Cargar una partida

- Se crea una nueva ventana (windowLoadGame) con configuraciones específicas de tamaño, ubicación y cierre.
- Se establece una imagen de fondo en la ventana utilizando un panel ImagePanel.
- Se obtienen las etiquetas de las partidas guardadas desde la base de datos mediante el método InteractionBD.getAllSavedGames() y se almacenan en un ArrayList llamado etiquetas.
- Se crea un panel para organizar los botones en forma de columna, con un fondo transparente.
- Se crean botones para cada etiqueta obtenida. Cada botón tiene un fondo y un color de texto específicos. Además, se añade un ActionListener a cada botón.
- Cuando se presiona un botón, se utiliza la etiqueta asociada para cargar la partida correspondiente desde la base de datos mediante InteractionBD.loadGame(etiqueta). Se verifica la integridad de los datos cargados y se actualizan las variables globales players y map.
- Se añaden los botones al panel y se establece un espacio entre ellos.
- Se configura un borde con espacio alrededor del panel.
- Se agrega el panel a la ventana en el centro.
- Finalmente, la ventana de carga se hace visible.

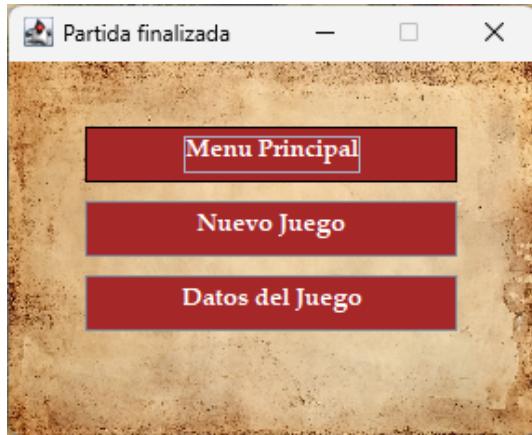


Figura 12: Menú al finalizar partida

- Se crea una nueva ventana (finishWindow) con configuraciones específicas de tamaño, ubicación y cierre.
- Se establece una imagen de fondo en la ventana utilizando un panel ImagePanel.
- Se crea un panel para organizar los botones en forma de columna, con un fondo transparente y una dimensión preferida.
- Se crean botones con etiquetas específicas y configuraciones de estilo. Cada botón tiene un ActionListener asociado.
- El primer botón ("Menu Principal") permite regresar al menú principal, cerrando la ventana actual y mostrando el menú principal.
- El segundo botón ("Nuevo Juego") cierra la ventana actual y muestra la ventana de configuración de jugadores para iniciar un nuevo juego.
- El tercer botón ("Datos del Juego") ejecuta una acción para exportar los datos del juego a un archivo de texto mediante ExportFile.fileWriter(players, map).
- Se añade espacio entre los botones y en los bordes del panel.
- El panel se agrega a la ventana en el centro.
- Finalmente, la ventana emergente se hace visible.

4.4. Base de Datos

Listing 1: Conexión a Base de Datos con Singleton

```
1  /*
2  Laboratorio Final - VideoGame
3  Autores:
4      - Eduardo Sebastian Stephan Portugal Portugal
5      - Hernan Andy Choquehuanca Zapana
6  */
7
8  //Imports para la conexión a la base de datos
9  import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.SQLException;
12
13 public class DataBase {
14     private static DataBase instance;
15
16     // La conexión a la base de datos
17     private Connection connection;
18
19     // Credenciales sobre la base de datos
20     private static final String JDBC_URL =
21         "jdbc:mysql://bzd2feukkimivqje2lb7-mysql.services.clever-cloud.com:3306/bzd2feukkimivqje2lb7";
22     private static final String USER = "uf812gj0y6ob1akl";
23     private static final String PASSWORD = "7nRZViLlzjURY9UTuZSd";
24
25     // Constructor privado para evitar instanciación directa
26     private DataBase() {
27         try {
28             // Cargar el driver de JDBC
29             Class.forName("com.mysql.cj.jdbc.Driver");
30
31             // Establecer la conexión a la base de datos
32             this.connection = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
33
34         } catch (ClassNotFoundException | SQLException e) {
35             e.printStackTrace();
36         }
37     }
38
39     // Método para obtener la instancia única
40     public static synchronized DataBase getInstance() {
41         if (instance == null) {
42             instance = new DataBase();
43         }
44         return instance;
45     }
46
47     // Método para obtener la conexión a la base de datos
48     public Connection getConnection() {
49         return this.connection;
50     }
}
```

- Se declara una variable estática instance de tipo DataBase para almacenar la única instancia de la clase.
- La clase tiene un campo privado connection que representa la conexión a la base de datos.
- Se definen constantes JDBC-URL, USER y PASSWORD que contienen la URL de conexión JDBC, el usuario y la contraseña para acceder a la base de datos.
- El constructor privado de la clase carga el driver de JDBC y establece la conexión a la base de datos usando las credenciales proporcionadas.
- Se implementa un método estático getInstance que devuelve la única instancia de la clase DataBase, creándola si aún no existe.
- Se proporciona un método getConnection que devuelve la conexión a la base de datos.
- El uso del patrón Singleton asegura que solo haya una instancia de la clase DataBase y proporciona un punto de acceso global a la conexión de la base de datos en toda la aplicación.

Listing 2: Guardar Partida

```

22 public class InteractionBD {
23     private static String fileName = "saveGame.txt"; //Nombre fijo del archivo
24
25     //Metodo para guardar la partida
26     public static void saveGame(String gameName, Object[] game){
27         //Escribe el arreglo de objetos
28         try (ObjectOutputStream fileOut = new ObjectOutputStream(new
29             FileOutputStream(fileName))) {
30             fileOut.writeObject(game);
31             System.out.println("Kingdom object saved to file: " + fileName);
32         } catch (IOException e) {
33             e.printStackTrace();
34         }
35         //Guarda el archivo en la base de datos
36         saveFileToDatabase(gameName);
37     }
38
39     //Metodo para guardar el archivo en la base de datos
40     private static void saveFileToDatabase(String gameName) {
41         try {
42             //Realiza la conexion a la base de datos e insercion del archivo
43             DataBase database = DataBase.getInstance();
44             Connection connection = database.getConnection();
45             String insertQuery = "INSERT INTO savegame (game_name, file_game) VALUES (?, ?)";
46
47             //Realiza la consulta
48             try (PreparedStatement preparedStatement =
49                 connection.prepareStatement(insertQuery)) {
50                 preparedStatement.setString(1, gameName);
51
52                 // Leer el contenido del archivo como un array de bytes
53                 byte[] fileContent = Files.readAllBytes(new File(fileName).toPath());
54
55                 // Almacenar el array de bytes en la base de datos
56                 preparedStatement.setBytes(2, fileContent);
57             }
58         }
59     }
60 }
```

```

56         //Realiza la consulta para guardar el archivo
57         preparedStatement.executeUpdate();
58         System.out.println("File saved to database.");
59     }
60 } catch (SQLException | IOException e) {
61     e.printStackTrace();
62 }
63 }
```

- La clase tiene un campo estático fileName que representa el nombre fijo del archivo en el cual se guarda la partida (saveGame.txt).
- El método público saveGame recibe el nombre de la partida (gameName) y un arreglo de objetos (game). Utiliza un ObjectOutputStream para escribir el arreglo de objetos en el archivo mencionado. Luego, llama al método privado saveFileToDatabase para guardar el archivo en la base de datos.
- El método privado saveFileToDatabase realiza las siguientes acciones:
 - Establece una conexión a la base de datos utilizando la clase DataBase.
 - Define una consulta SQL para insertar el nombre de la partida y el contenido del archivo en la tabla savegame.
 - Utiliza un PreparedStatement para ejecutar la consulta, asignando el nombre de la partida y el contenido del archivo al statement.
 - Lee el contenido del archivo como un array de bytes y lo almacena en la base de datos como un campo binario (BLOB).
 - Ejecuta la consulta SQL para guardar el archivo en la base de datos.

Listing 3: Cargar Partida

```

65 //Metodo para obtener las partidas guardadas
66 public static ArrayList<String> getAllSavedGames(){
67     ArrayList<String> gameNames = new ArrayList<>();
68     //Consulta a la base de datos acerca de los juegos que se encuentren guardados
69     try {
70         DataBase database = DataBase.getInstance();
71         Connection connection = database.getConnection();
72
73         String query = "SELECT game_name FROM savegame";
74         try (PreparedStatement preparedStatement = connection.prepareStatement(query);
75             ResultSet resultSet = preparedStatement.executeQuery()) {
76
77             while (resultSet.next()) {
78                 String gameName = resultSet.getString("game_name");
79                 gameNames.add(gameName);
80             }
81         }
82     } catch (SQLException e) {
83         e.printStackTrace();
84     }
85
86     return gameNames;
87 }
```

```

88
89     //Metodo para cargar una partida
90     //Devuelve un arreglo de objetos obtenidos de la base de datos
91     public static Object[] loadGame(String gameName){
92         Object[] game = null;
93         //Realiza la conexión a la base de datos y la consulta
94         try {
95             DataBase database = DataBase.getInstance();
96             Connection connection = database.getConnection();
97
98             String query = "SELECT file_game FROM savegame WHERE game_name = ?";
99             try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {
100                 preparedStatement.setString(1, gameName);
101                 try (ResultSet resultSet = preparedStatement.executeQuery()) {
102                     if (resultSet.next()) {
103                         // Obtener el flujo de bytes desde la base de datos
104                         InputStream fileContentStream = resultSet.getBinaryStream("file_game");
105
106                         // Deserializar el objeto Kingdom desde el flujo de bytes
107                         try (ObjectInputStream fileIn = new
108                             ObjectInputStream(fileContentStream)) {
109                             game = (Object[]) fileIn.readObject();
110                         }
111                     }
112                 }
113             } catch (SQLException | IOException | ClassNotFoundException e) {
114                 e.printStackTrace();
115             }
116
117             return game;
118         }
119     }

```

- getAllSavedGames:
 - Este método obtiene todos los nombres de las partidas guardadas en la base de datos.
 - Realiza una consulta SQL para seleccionar los nombres de las partidas (SELECT game_name FROM savegame).
- Retorna el ArrayList que contiene los nombres de todas las partidas guardadas.
- loadGame:
 - Este método carga una partida desde la base de datos según el nombre de la partida proporcionado.
 - Realiza una consulta SQL para seleccionar el contenido de la partida (SELECT file_game FROM savegame WHERE ?). Utiliza un PreparedStatement para establecer el nombre de la partida en la consulta.
- Lee el resultado de la consulta, que contiene el contenido de la partida como un flujo de bytes.
- Deserializa el objeto desde el flujo de bytes utilizando un ObjectInputStream para obtener el arreglo de objetos (Object[]).
- Retorna el arreglo de objetos que representa la partida cargada.

Listing 4: Ingresar y Registrar Usuario

```

121 // Método para verificar si el usuario est registrado
122 public static boolean authenticateUser(String nickName, String password) {
123     // Definir la consulta SQL
124     String query = "SELECT * FROM users WHERE nick_name = ? AND password = ?";
125     try {
126         DataBase database = DataBase.getInstance();
127         Connection connection = database.getConnection();
128         PreparedStatement preparedStatement = connection.prepareStatement(query);
129         // Establecer los parmetros en la consulta
130         preparedStatement.setString(1, nickName);
131         preparedStatement.setString(2, password);
132
133         // Ejecutar la consulta
134         try (ResultSet resultSet = preparedStatement.executeQuery()) {
135             // Si hay al menos una fila en el resultado, el usuario est registrado
136             return resultSet.next();
137         }
138     } catch (SQLException e) {
139         e.printStackTrace();
140     }
141
142     // En caso de error o si no se encuentra el usuario, retornar false
143     return false;
144 }
145
146 // Método para registrar un nuevo usuario
147 public static boolean registerUser(String nickName, String password) {
148     // Verificar si ya existe un usuario con el mismo nick_name
149     if (userExists(nickName)) {
150         System.out.println("Ya existe un usuario con el mismo nick_name.");
151         return false;
152     }
153
154     // Si no existe, proceder con el registro
155     String insertQuery = "INSERT INTO users (nick_name, password) VALUES (?, ?)";
156
157     try {
158         DataBase database = DataBase.getInstance();
159         Connection connection = database.getConnection();
160         PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);
161
162         // Establecer los parmetros en la consulta de inserción
163         preparedStatement.setString(1, nickName);
164         preparedStatement.setString(2, password);
165
166         // Ejecutar la consulta de inserción
167         int rowsAffected = preparedStatement.executeUpdate();
168
169         // Si al menos una fila fue afectada, el registro fue exitoso
170         return rowsAffected > 0;
171
172     } catch (SQLException e) {
173         e.printStackTrace();
174     }
175 }
```

```

176     // En caso de error, retornar false
177     return false;
178 }
179
180 // Método auxiliar para verificar si ya existe un usuario con el mismo nick_name
181 private static boolean userExists(String nickName) {
182     String query = "SELECT * FROM users WHERE nick_name = ?";
183
184     try {
185         DataBase database = DataBase.getInstance();
186         Connection connection = database.getConnection();
187         PreparedStatement preparedStatement = connection.prepareStatement(query);
188
189         // Establecer el parmetro en la consulta de selección
190         preparedStatement.setString(1, nickName);
191
192         // Ejecutar la consulta de selección
193         return preparedStatement.executeQuery().next();
194
195     } catch (SQLException e) {
196         e.printStackTrace();
197     }
198
199     // En caso de error, asumir que el usuario existe
200     return true;
201 }
202 }
```

■ authenticateUser:

- Este método verifica si un usuario está registrado en el sistema.
- Utiliza una consulta SQL para seleccionar filas de la tabla “users” que coincidan con el nick_name y la contraseña proporcionados.
- Retorna true si al menos una fila es devuelta por la consulta, indicando que el usuario está registrado; de lo contrario, retorna false.
- Maneja excepciones de SQL, imprimiendo la traza de la excepción en caso de error.

■ registerUser:

- Este método registra un nuevo usuario en el sistema.
- Verifica primero si ya existe un usuario con el mismo nick_name utilizando el método userExists.
- Si el usuario no existe, procede a realizar una consulta de inserción SQL para agregar un nuevo usuario a la tabla “users”.
- Retorna true si al menos una fila fue afectada por la inserción, indicando que el registro fue exitoso; de lo contrario, retorna false.
- Maneja excepciones de SQL, imprimiendo la traza de la excepción en caso de error.

■ userExists:

- Este método es un auxiliar utilizado por registerUser para verificar si ya existe un usuario con el mismo nick_name.

- Utiliza una consulta SQL para seleccionar filas de la tabla "users" que coincidan con el nick_name proporcionado.
- Retorna true si al menos una fila es devuelta por la consulta, indicando que el usuario ya existe; de lo contrario, retorna false.
- Maneja excepciones de SQL, imprimiendo la traza de la excepción en caso de error.

4.5. Exportar datos del VideoJuego

Listing 5: Exportar Archivo con Datos del VideoJuego

```

1  /*
2  Laboratorio Final - VideoGame
3  Autores:
4      - Eduardo Sebastian Stephan Portugal Portugal
5      - Hernan Andy Choquehuana Zapana
6  */
7
8 //Imports para la conexion a la base de datos
9 import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.SQLException;
12
13 public class DataBase {
14     private static DataBase instance;
15
16     // La conexin a la base de datos
17     private Connection connection;
18
19     // Credenciales sobre la base de datos
20     private static final String JDBC_URL =
21         "jdbc:mysql://bzd2feukkimivqje2lb7-mysql.services.clever-cloud.com:3306/bzd2feukkimivqje2lb7";
22     private static final String USER = "uf812gj0y6ob1akl";
23     private static final String PASSWORD = "7nRZViLlzjURY9UTuZSd";
24
25     // Constructor privado para evitar instanciacin directa
26     private DataBase() {
27         try {
28             // Cargar el driver de JDBC
29             Class.forName("com.mysql.cj.jdbc.Driver");
30
31             // Establecer la conexin a la base de datos
32             this.connection = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
33
34         } catch (ClassNotFoundException | SQLException e) {
35             e.printStackTrace();
36         }
37     }
38
39     // Metodo para obtener la instance nica
40     public static synchronized DataBase getInstance() {
41         if (instance == null) {
42             instance = new DataBase();
43         }
44         return instance;
45     }

```

```

45
46     // Método para obtener la conexión a la base de datos
47     public Connection getConnection() {
48         return this.connection;
49     }
50 }
```

- La función fileWriter recibe un ArrayList de jugadores (game) y un objeto Mapa (map) que representan el estado de una partida del juego "Shadow of the Kingdoms".
- Crea un archivo de texto con el nombre "DataGame.txt" utilizando un objeto PrintWriter para escribir en él.
- Imprime información sobre la partida, incluyendo los nombres de los reinos que participan, los ejércitos de cada reino, los soldados de cada ejército y detalles del mapa.
- Itera sobre los jugadores en el ArrayList y escribe información sobre cada uno, incluyendo detalles sobre su reino (MyKingdom) utilizando el método dataFile() del reino.
- Cierra el objeto PrintWriter después de completar la escritura en el archivo.
- Imprime un mensaje en la consola indicando que la escritura en el archivo fue exitosa.
- Utiliza la clase Desktop para abrir el archivo recién creado con la aplicación predeterminada para archivos en el sistema.
- Maneja excepciones de IO, imprimiendo mensajes de error en caso de problemas al escribir o abrir el archivo.

Listing 6: Ejemplo de los datos del VideoJuego guardados

```
Shadow of the kingdoms
```

```
Inglaterra VS Francia
```

```
Mapa: Territorio: campo abierto Cantidad de ejércitos: 7
```

```
Player 1: NickName: Reino: Inglaterra Color: red
```

```
Reino: Inglaterra
```

```
Ejército: Ejército-1 Activo: true Vida total: 38 Soldados creados: 4 Soldados vivos: 4
Espadachines: 2 Arqueros: 0 Caballeros: 1 Lanceros: 1 Posición: A5
```

```
Soldado: Espadachin_Real Nivel de ataque: 10 Nivel de defensa: 8 Nivel de vida: 12 Vida
actual: 12 Velocidad: 0 Actitud: neutra Vivo: true Reino: 1 Ejército: 1 Posición:
...
```

- En el archivo creado se muestra un resumen de todos los datos del VideoJuego, describiendo todos los atributos de los Players, de sus Reinos, Ejércitos y Soldados.

Listing 7: Commits del archivo VideoGame.java

```
$ git commit -m "Clase VideoGame"
$ git commit -m "Clase ExportFile"
$ git commit -m "Clase DataBase"
$ git commit -m "Clase Interaction BD"
$ git commit -m "Imagenes"
$ git commit -m "Librerias"
$ git commit -m "VideoGame.jar"
```

- Estos son los commits más importantes realizados durante la creación de VideoGame.java, puesto que los métodos que se registran, resultan ser los determinantes para el funcionamiento y estructura gráfica principal de programa.
- Código del commit 1: 1e52bd271e5046b38f4117f18c3fe504a0991c03
- Código del commit 2: e36bec829e2cb039555d4edecf2ada035b070677
- Código del commit 3: f1597abeff86b1596bc5d8c149455034bf9e4ec3
- Código del commit 4: cfd6b30795d3b294d9819be57ee0f16d1ec4e8a2
- Código del commit 5: 7d9356acb9ff666151f9ba94ce7c29d868d3ffe2
- Código del commit 6: 20b3374f9a992c8feb61dead9502dc68f77f09e2
- Código del commit 7: 3cc881520dbbca53c54e86dc303ce184df91a763

4.6. Estructura de laboratorio Final

- El contenido que se entrega en este laboratorio es el siguiente:

```
labFinal/
|--- Archer.class
|--- Archer.java
|--- Army.class
|--- Army.java
|--- Arrows.class
|--- Arrows.java
|--- Coalition.class
|--- Coalition.java
|--- DataBase.class
|--- DataBase.java
|--- DataGame.txt
|--- ExportFile.class
|--- ExportFile.java
|--- Hatchets.class
|--- Hatchets.java
|--- InteractionBD.class
|--- InteractionBD.java
|--- Javelins.class
|--- Javelins.java
|--- Kingdom.class
|--- Kingdom.java
|--- Knight.class
|--- Knight.java
|--- KnightFranco.class
|--- KnightFranco.java
|--- KnightMoro.class
|--- KnightMoro.java
|--- Knives.class
|--- Knives.java
|--- MANIFEST.MF
|--- Map$1.class
|--- Map$2.class
|--- Map.class
|--- Map.java
|--- Military.class
|--- Military.java
|--- Player.class
|--- Player.java
|--- saveGame.txt
|--- Soldier.class
|--- Soldier.java
|--- Spearman.class
|--- Spearman.java
|--- Spears.class
|--- Spears.java
|--- Swordsman.class
|--- Swordsman.java
|--- SwordsmanConqueror.class
|--- SwordsmanConqueror.java
|--- SwordsmanRoyal.class
```

```
|--- SwordsmanRoyal.java
|--- SwordsmanTeutonic.class
|--- SwordsmanTeutonic.java
|--- VideoGame$1.class
|--- VideoGame$2.class
|--- VideoGame$3.class
|--- VideoGame$4.class
|--- VideoGame$5.class
|--- VideoGame$6.class
|--- VideoGame$ImagePanel.class
|--- VideoGame.class
|--- VideoGame.jar
|--- VideoGame.java
|--- Weapons.class
|--- Weapons.java
|--- lib
    |--- mysql-connector-j-8.3.0.jar
|--- src
    |--- ...
    |--- swordsman_yellow.png
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplió con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

Puntos	Nivel			
	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

	Contenido y demostración	Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	

6. Referencias

<https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>
<https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html>
<https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html>
<https://developer.oracle.com/es/learn/java-and-databases.html>
<https://docs.oracle.com/javase/8/docs/api/java/io/File.html>