

## Informe de Laboratorio Final 24

Nota

Estudiante	Escuela	Asignatura
Quispe Arratea Alexandra Raquel	Escuela Profesional de Ingeniería de Sistemas	FP-II Semestre: II

Laboratorio	Tema	Duración
24		24 horas aprox.

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 15 Enero 2024	Al 2 Febrero 2024

### 1. Tarea

- Cree una versión del videojuego de estrategia usando componentes básicos GUI: Etiquetas, botones, cuadros de texto, JOptionPane, Color. Además, utilizar componentes avanzados GUI: Layouts, JPanel, áreas de texto, checkbox, botones de radio y combobox. Considerar nivel estratégico y táctico. Considerar hasta las unidades especiales de los reinos. Hacerlo iterativo.

### 2. Equipos, materiales y temas utilizados

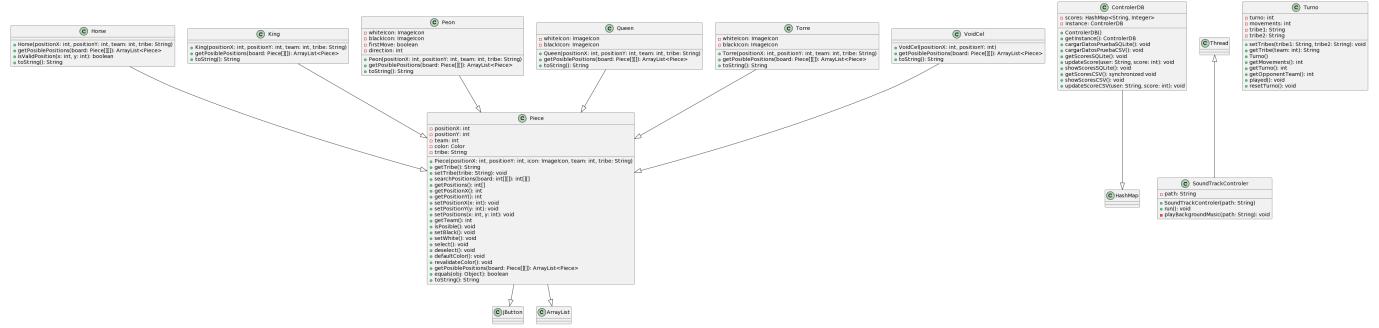
- Sistemas Operativos Ubuntu y Windows
- NVIM v0.6.1, VSCode, Eclipse
- OpenJDK 64-Bits 11, 17, 18
- Git 2.34.1 y versiones posteriores
- Cuentas en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Algoritmos de ordenamiento y búsqueda
- Modelizado de algoritmos

### 3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/JhonatanDczel/FP2-trabajo-final.git>

## 4. Proyecto LabFinal

- Generamos diagrama UML del proyecto :



## 5. Contextualización del Juego

En este proyecto, hemos tomado la base de un juego de estrategia con soldados y lo hemos adaptado para crear un emocionante juego de ajedrez con funciones más dinámicas. Esta adaptación sigue aplicando los conceptos aprendidos durante el curso y busca proporcionar a los jugadores una experiencia única en el mundo del ajedrez.

Hemos transformado el juego original para que sea un enfrentamiento uno contra uno, donde los jugadores se dividen en cuatro pueblos, cada uno con características especiales, bonificaciones y penalizaciones. A continuación, presentamos cada uno de los pueblos, detallando sus características únicas.

### 5.1. Macondo

Macondo es un pueblo mágico y aislado, donde ocurren sucesos fantásticos e insólitos. Sus habitantes son soñadores, curiosos y creativos, pero también sufren de olvido. Las fichas de ajedrez podrían tener un distintivo de un gorro de mago.

**Bonificación de Portal Mágico:** Puede intercambiar de lugar dos fichas aliadas una vez por partida.

**Penalización de Olvido:** Si Macondo no utiliza su bonificación de portal mágico antes de alcanzar la mitad de la partida, pierde 5 puntos de vida por haber olvidado sus habilidades mágicas.



## 5.2. Comala

Comala es un pueblo fantasmal y desolado, donde los muertos conviven con los vivos. Sus habitantes son silenciosos, melancólicos y resignados, pero también buscan la redención y el perdón. Las fichas de ajedrez podrían tener un distintivo de una vela, que simboliza la fe y el sufrimiento.

**Bonificación de Fantasma:** Puede esquivar un ataque enemigo, reduciendo el daño recibido a la mitad, una vez por turno.

**Penalización de Soledad:** Pierde 15 puntos de vida si está completamente rodeado por fichas enemigas.



## 5.3. Rivendel

Rivendel es un reino élfico y hermoso, donde reina la armonía y la sabiduría. Sus habitantes son ágiles, bellos y sabios, pero también nostálgicos y orgullosos. Las fichas de ajedrez podrían tener un distintivo de una corona de oro, que simboliza la luz y la elegancia.

**Bonificación de Luz Élfica:** Puede realizar un movimiento especial (moverse donde sea) una vez por partida que afecta a una ficha enemiga cercana.

**Penalización de Nostalgia:** La añoranza de Rivendel lo distrae, perdiendo 20 puntos de ataque si se aleja más de tres casillas de su posición inicial.



## 5.4. Dorne

Dorne es un reino sureño y exótico, donde se celebra la diversidad y la libertad. Sus habitantes son carismáticos, seductores y venenosos, pero también rebeldes e indomables. Las fichas de ajedrez podrían tener un distintivo de tatuajes, que simboliza el exotismo.

**Bonificación de Solidaridad Exótica:** Puede transferir 10 puntos de defensa adicional a una ficha aliada una vez por turno.

**Penalización de Rebeldía:** La rebeldía de Dorne aumenta su fuerza, pero también su vulnerabilidad. Obtendrá un bonus de ataque aleatorio (entre 10 y 30 puntos), pero perderá 15 puntos de defensa.



## 6. Desarrollo de clases

### 6.1. Clase BattleWindow

La clase `BattleWindow` representa la ventana de la batalla en el juego. A continuación, se presenta un resumen de su estructura y funcionamiento:

#### 6.1.1. Atributos:

- `attacker`: Pieza atacante en la batalla.

Listing 1: Atributo `attacker` en la clase `BattleWindow`

```
private Piece attacker;
```

- `defender`: Pieza defensora en la batalla.

Listing 2: Atributo `defender` en la clase `BattleWindow`

```
private Piece defender;
```

#### 6.1.2. Métodos Relevantes:

- `BattleWindow(Piece attacker, Piece defender)`: Constructor que inicializa la ventana de la batalla con las piezas proporcionadas.

Listing 3: Constructor `BattleWindow` que recibe las piezas atacante y defensora

```

public BattleWindow(Piece attacker, Piece defender) {
    this.attacker = attacker;
    this.defender = defender;

    // Configuración de la interfaz gráfica y creación de paneles
    // ...

    // Agrega las imágenes a los paneles
    attackerPanel.add(attackerLabel);
    defenderPanel.add(defenderLabel);

    // Configuración y visualización de la ventana
    setSize(300, 300);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setVisible(true);
}

```

- **BattleBar:** Clase interna que representa la barra de la batalla y calcula el resultado de la misma.

Listing 4: Clase interna BattleBar en la clase BattleWindow

```

private class BattleBar extends JPanel {
    // Implementación de la clase interna
    // ...
}

```

- **calculateBattleResult():** Método que debe ser implementado para calcular el resultado de la batalla. Retorna un valor entre 0 y 100 representando el porcentaje de éxito del atacante.

Listing 5: Método calculateBattleResult en la clase BattleBar

```

private int calculateBattleResult() {
    // Método que debes implementar para calcular el resultado de la batalla
    // Debes retornar un valor entre 0 y 100 representando el porcentaje de éxito del
    // atacante
    return 50; // Ejemplo: Empate
}

```

Esta clase proporciona una interfaz gráfica para visualizar la batalla entre dos piezas en el juego. La barra horizontal en el centro de la ventana representa visualmente el resultado de la batalla.

## 6.2. Clase Board

La clase **Board** representa el tablero del juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

### 6.2.1. Atributos:

- **board:** Matriz que representa el tablero de juego.

Listing 6: Atributo board en la clase Board

```

private Piece[][] board = new Piece[8][8];

```

- **turno**: Objeto que lleva el control del turno actual.

Listing 7: Atributo **turno** en la clase **Board**

```
private Turno turno = new Turno();
```

- **finishBtn**: Pieza que representa la posición de inicio del movimiento.

Listing 8: Atributo **finishBtn** en la clase **Board**

```
private Piece finishBtn = null;
```

- **possiblePositions**: Lista de posiciones posibles para una pieza.

Listing 9: Atributo **possiblePositions** en la clase **Board**

```
private ArrayList<Piece> possiblePositions;
```

- **box1, box2**: Paneles que contienen componentes del juego.

Listing 10: Atributos **box1** y **box2** en la clase **Board**

```
private JPanel box1;  
private JPanel box2;
```

- **btnSave, btnRecover, btnNewPPlay**: Botones para guardar, recuperar y reiniciar el juego.

Listing 11: Atributos **btnSave**, **btnRecover**, y **btnNewPPlay** en la clase **Board**

```
private JButton btnSave = new JButton("SAVE GAME");  
private JButton btnRecover = new JButton("RECOVER GAME");  
private JButton btnNewPPlay = new JButton("NEW PLAY");
```

### 6.2.2. Métodos Relevantes:

- **Board()**: Constructor que inicia un nuevo juego de ajedrez.

Listing 12: Constructor **Board** en la clase **Board**

```
public Board() {  
    // Implementación del constructor  
    // ...  
}
```

- **ListenerBtn**: Clase interna que implementa **ActionListener** para manejar eventos de botones.

Listing 13: Clase interna **ListenerBtn** en la clase **Board**

```
private class ListenerBtn implements ActionListener {  
    // Implementación de la clase interna  
    // ...  
}
```

- **resetGame()**: Reinicia el juego con nuevas tribus.

Listing 14: Método `resetGame` en la clase `Board`

```
public void resetGame() {
    // Implementación del reinicio del juego
    // ...
}
```

- `saveGame()`: Guarda el estado actual del juego en un archivo binario.

Listing 15: Método `saveGame` en la clase `Board`

```
public void saveGame() {
    // Implementación del guardado del juego
    // ...
}
```

- `recoverGame()`: Recupera un juego guardado anteriormente.

Listing 16: Método `recoverGame` en la clase `Board`

```
public void recoverGame() {
    // Implementación de la recuperación del juego
    // ...
}
```

- `removeBoard()`: Elimina las piezas del tablero.

Listing 17: Método `removeBoard` en la clase `Board`

```
public void removeBoard() {
    // Implementación de la eliminación de las piezas del tablero
    // ...
}
```

- `reFill()`: Vuelve a llenar el tablero con las piezas.

Listing 18: Método `reFill` en la clase `Board`

```
public void reFill() {
    // Implementación del relleno del tablero
    // ...
}
```

La clase `Board` proporciona la lógica y la interfaz para el juego de ajedrez, gestionando el tablero, las piezas y las interacciones entre los jugadores.

### 6.3. Clase Horse

La clase `Horse` representa la pieza de caballo en el juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

#### 6.3.1. Atributos:

- `whiteIcon`, `blackIcon`: Iconos para la pieza de caballo en los equipos blanco y negro.

Listing 19: Atributos `whiteIcon` y `blackIcon` en la clase `Horse`

```
private static ImageIcon whiteIcon = new ImageIcon("./assets/whiteHorse.png");
private static ImageIcon blackIcon = new ImageIcon("./assets/blackHorse.png");
```

### 6.3.2. Métodos Relevantes:

- `Horse(int positionX, int positionY, int team, String tribe)`: Constructor que inicializa la pieza de caballo con la posición, equipo y tribu correspondientes.

Listing 20: Constructor `Horse` en la clase `Horse`

```
public Horse(int positionX, int positionY, int team, String tribe) {
    // Implementación del constructor
    // ...
}
```

- `getPossiblePositions(Piece[][] board)`: Obtiene las posiciones posibles a las que puede moverse la pieza de caballo.

Listing 21: Método `getPossiblePositions` en la clase `Horse`

```
public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
    // Implementación del método
    // ...
}
```

- `isValidPosition(int x, int y)`: Verifica si una posición dada en el tablero es válida.

Listing 22: Método `isValidPosition` en la clase `Horse`

```
private boolean isValidPosition(int x, int y) {
    // Implementación del método
    // ...
}
```

- `toString()`: Devuelve una representación en cadena de la pieza de caballo.

Listing 23: Método `toString` en la clase `Horse`

```
public String toString(){
    return "Caballo "+super.toString();
}
```

La clase `Horse` encapsula la lógica específica del caballo en el juego de ajedrez, determinando sus movimientos posibles y proporcionando una representación en cadena para fines de visualización.

### 6.4. Clase King

La clase `King` representa la pieza de rey en el juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

#### 6.4.1. Atributos:

- `whiteIcon, blackIcon`: Iconos para la pieza de rey en los equipos blanco y negro.

Listing 24: Atributos `whiteIcon` y `blackIcon` en la clase King

```
private static ImageIcon whiteIcon = new ImageIcon("./assets/whiteKing.png");
private static ImageIcon blackIcon = new ImageIcon("./assets/blackKing.png");
```

#### 6.4.2. Métodos Relevantes:

- `King(int positionX, int positionY, int team, String tribe)`: Constructor que inicializa la pieza de rey con la posición, equipo y tribu correspondientes.

Listing 25: Constructor King en la clase King

```
public King(int positionX, int positionY, int team, String tribe) {
    // Implementación del constructor
    // ...
}
```

- `getPossiblePositions(Piece[][] board)`: Obtiene las posiciones posibles a las que puede moverse la pieza de rey.

Listing 26: Método `getPossiblePositions` en la clase King

```
public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
    // Implementación del método
    // ...
}
```

- `toString()`: Devuelve una representación en cadena de la pieza de rey.

Listing 27: Método `toString` en la clase King

```
public String toString(){
    return "Rey "+super.toString();
}
```

La clase King encapsula la lógica específica del rey en el juego de ajedrez, determinando sus movimientos posibles y proporcionando una representación en cadena para fines de visualización.

### 6.5. Clase Peon

La clase Peon representa la pieza de peón en el juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

#### 6.5.1. Atributos:

- `whiteIcon, blackIcon`: Iconos para la pieza de peón en los equipos blanco y negro.

Listing 28: Atributos `whiteIcon` y `blackIcon` en la clase Peon

```
private static ImageIcon whiteIcon = new ImageIcon("./assets/whitePeon.png");
private static ImageIcon blackIcon = new ImageIcon("./assets/blackPeon.png");
```

- **firstMove**: Indica si es el primer movimiento del peón.

Listing 29: Atributo **firstMove** en la clase Peon

```
private boolean firstMove = true;
```

- **direction**: Dirección del movimiento del peón según el equipo.

Listing 30: Atributo **direction** en la clase Peon

```
private int direction;
```

#### 6.5.2. Métodos Relevantes:

- **Peon(int positionX, int positionY, int team, String tribe)**: Constructor que inicializa la pieza de peón con la posición, equipo y tribu correspondientes.

Listing 31: Constructor Peon en la clase Peon

```
public Peon(int positionX, int positionY, int team, String tribe) {
    // Implementación del constructor
    // ...
}
```

- **getPossiblePositions(Piece[][] board)**: Obtiene las posiciones posibles a las que puede moverse la pieza de peón.

Listing 32: Método **getPossiblePositions** en la clase Peon

```
public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
    // Implementación del método
    // ...
}
```

- **toString()**: Devuelve una representación en cadena de la pieza de peón.

Listing 33: Método **toString** en la clase Peon

```
public String toString(){
    return "Peon "+super.toString();
}
```

La clase **Peon** encapsula la lógica específica del peón en el juego de ajedrez, incluyendo su primer movimiento especial y la determinación de sus movimientos posibles.“

#### 6.6. Clase Queen

La clase **Queen** representa la pieza de reina en el juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

##### 6.6.1. Atributos:

- **whiteIcon, blackIcon**: Iconos para la pieza de reina en los equipos blanco y negro.

Listing 34: Atributos `whiteIcon` y `blackIcon` en la clase `Queen`

```
private static ImageIcon whiteIcon = new ImageIcon("./assets/whiteQueen.png");
private static ImageIcon blackIcon = new ImageIcon("./assets/blackQueen.png");
```

#### 6.6.2. Métodos Relevantes:

- `Queen(int positionX, int positionY, int team, String tribe)`: Constructor que inicializa la pieza de reina con la posición, equipo y tribu correspondientes.

Listing 35: Constructor `Queen` en la clase `Queen`

```
public Queen(int positionX, int positionY, int team, String tribe) {
    // Implementación del constructor
    // ...
}
```

- `getPossiblePositions(Piece[][] board)`: Obtiene las posiciones posibles a las que puede moverse la pieza de reina.

Listing 36: Método `getPossiblePositions` en la clase `Queen`

```
public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
    // Implementación del método
    // ...
}
```

- `toString()`: Devuelve una representación en cadena de la pieza de reina.

Listing 37: Método `toString` en la clase `Queen`

```
public String toString(){
    return "Reina "+super.toString();
}
```

La clase `Queen` encapsula la lógica específica de la reina en el juego de ajedrez, incluyendo la determinación de sus movimientos posibles en diferentes direcciones.“

### 6.7. Clase Torre

La clase `Torre` representa la pieza de torre en el juego de ajedrez. A continuación, se presenta un resumen de su estructura y funcionamiento:

#### 6.7.1. Atributos:

- `whiteIcon`, `blackIcon`: Iconos para la pieza de torre en los equipos blanco y negro.

Listing 38: Atributos `whiteIcon` y `blackIcon` en la clase `Torre`

```
private static ImageIcon whiteIcon = new ImageIcon("./assets/whiteTorre.png");
private static ImageIcon blackIcon = new ImageIcon("./assets/blackTorre.png");
```

### 6.7.2. Métodos Relevantes:

- **Torre(int positionX, int positionY, int team, String tribe):** Constructor que inicializa la pieza de torre con la posición, equipo y tribu correspondientes.

Listing 39: Constructor Torre en la clase Torre

```
public Torre(int positionX, int positionY, int team, String tribe) {
    // Implementación del constructor
    // ...
}
```

- **getPossiblePositions(Piece[][] board):** Obtiene las posiciones posibles a las que puede moverse la pieza de torre.

Listing 40: Método getPossiblePositions en la clase Torre

```
public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
    // Implementación del método
    // ...
}
```

- **toString():** Devuelve una representación en cadena de la pieza de torre.

Listing 41: Método toString en la clase Torre

```
public String toString(){
    return "Torre "+super.toString();
}
```

La clase **Torre** encapsula la lógica específica de la torre en el juego de ajedrez, incluyendo la determinación de sus movimientos posibles en direcciones vertical y horizontal.“

### 6.8. Clase VoidCel:

A continuación se presenta el código de la clase que representa una celda vacía en un tablero de ajedrez en Java.

Listing 42: Clase VoidCel

```
import java.awt.Cursor;
import java.util.ArrayList;

public class VoidCel extends Piece {
    public VoidCel(int positionX, int positionY) {
        super(positionX, positionY, null, -1, "none");
        setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    };

    public ArrayList<Piece> getPossiblePositions(Piece[][] board) {
        return new ArrayList<Piece>();
    };
    public String toString(){
        return "Vacio "+super.toString();
    }
}
```

## 6.9. Base de Datos

Código SQL para crear una base de datos con una tabla de usuarios y datos de ejemplo:

```
-- Crear la base de datos
CREATE DATABASE IF NOT EXISTS fp2_23b;

-- Usar la base de datos
USE fp2_23b;

-- Crear la tabla de usuarios
CREATE TABLE IF NOT EXISTS usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario VARCHAR(255) NOT NULL,
    puntuacion INT NOT NULL
);

-- Insertar datos de ejemplo
INSERT INTO usuarios (usuario, puntuacion) VALUES
('Pepito', 10),
('Juanito', 150),
('Reginald', 80);
```

## 6.10. Pista de Sonido

A continuación, se proporciona el código de un controlador de pista de sonido en Java, diseñado para reproducir música en segundo plano mediante el uso de la biblioteca javax.sound.sampled. Este controlador no solo ofrece funcionalidades estándar para la reproducción de sonido, sino que también implementa técnicas de programación paralela para optimizar el rendimiento y aprovechar eficientemente los recursos del sistema.

Para obtener puntos adicionales, hemos abordado este problema soundtrack . En lugar de reproducirlo como fondo, lo hemos implementado para ejecutarse de forma paralela. Esta solución permite que el soundtrack y el juego se desarrollen simultáneamente, evitando que el programa espere a que el soundtrack finalice para concluir la partida.

### 6.10.1. Clase SoundTrackController:

Listing 43: Clase SoundTrackController

```
import javax.sound.sampled.*;
import java.io.File;

public class SoundTrackController extends Thread {
    private String path;

    public SoundTrackController(String path) {
        this.path = path;
    }

    @Override
    public void run() {
        playBackgroundMusic(path);
    }
}
```

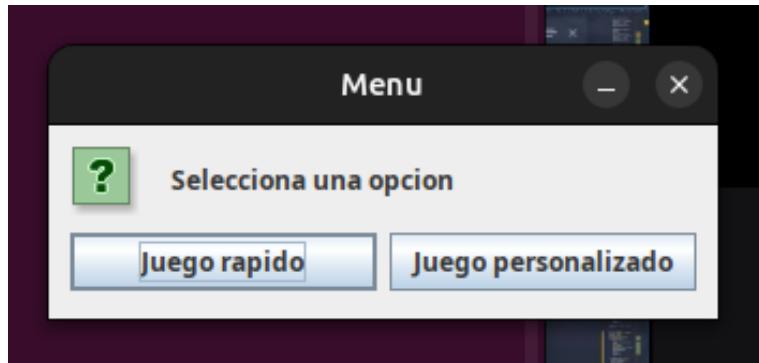
```

private void playBackgroundMusic(String path) {
    try {
        Clip clip = AudioSystem.getClip();
        AudioInputStream inputStream = AudioSystem.getAudioInputStream(new File(path));
        clip.open(inputStream);
        clip.start();
        clip.loop(Clip.LOOP_CONTINUOUSLY);
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}

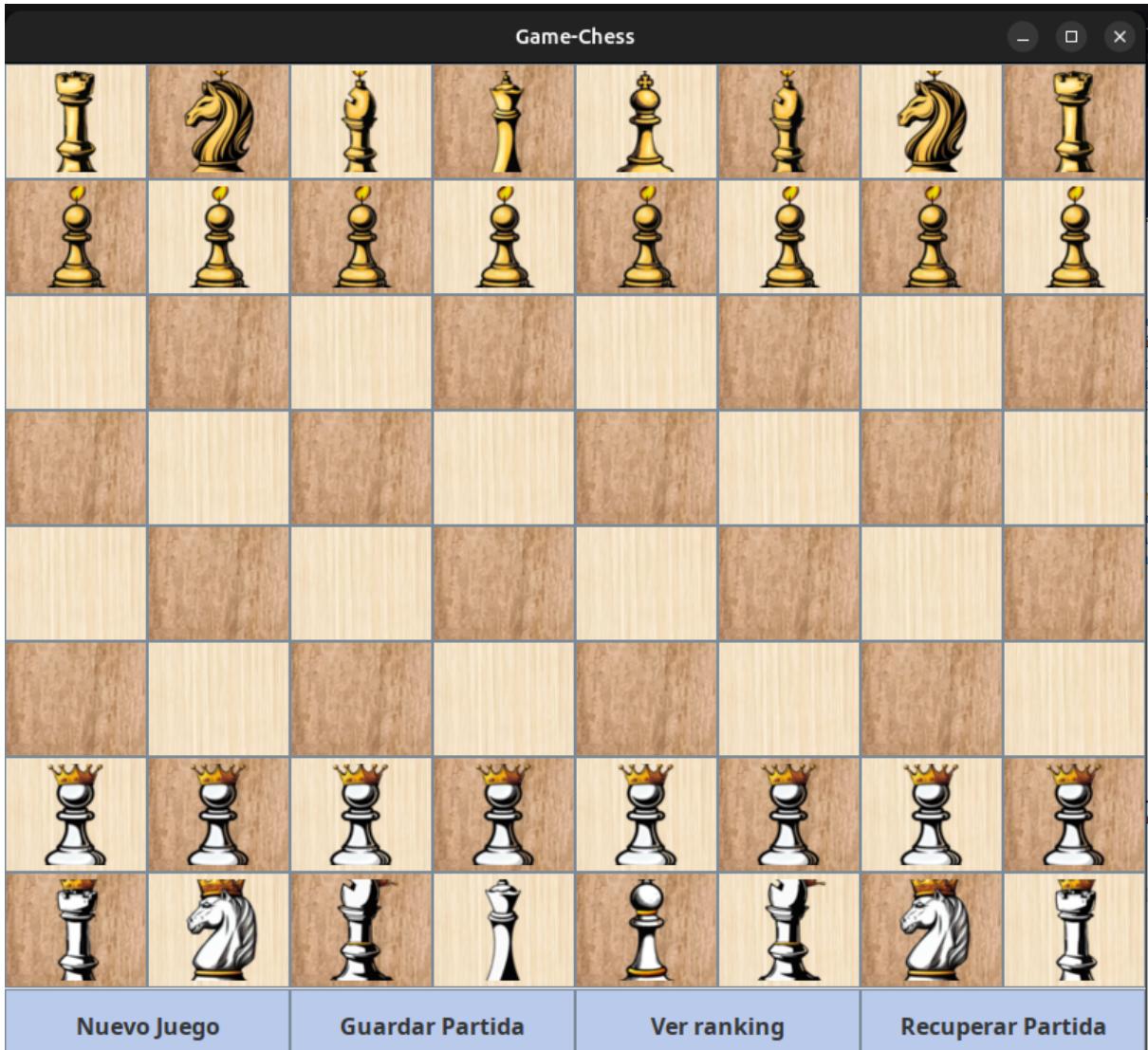
```

## 7. Ejecucion del Juego

- Primero, se presenta un menú con dos opciones: "Juego Personalizado" y "Juego Rápido". En el caso de seleccionar "Juego Personalizado", el sistema redirige al usuario a una interfaz de selección de pueblos, ofreciendo la posibilidad de elegir entre los 4 disponibles. Por otro lado, al optar por "Juego Rápido", el sistema realiza la selección aleatoria de 2 pueblos y procede a iniciar el juego con dichas elecciones.



- Estamos a punto de crear un juego personalizado, donde seleccionaremos a los pueblos de Mordor y Rivendel para participar.



- La ejecución del juego ha comenzado, y nos encontramos en un tablero con una disposición similar al de un tablero de ajedrez. Ahora exploraremos las opciones disponibles, las cuales incluyen la capacidad de guardar la partida para su recuperación posterior, acceder al ranking de jugadores y comenzar un nuevo juego.



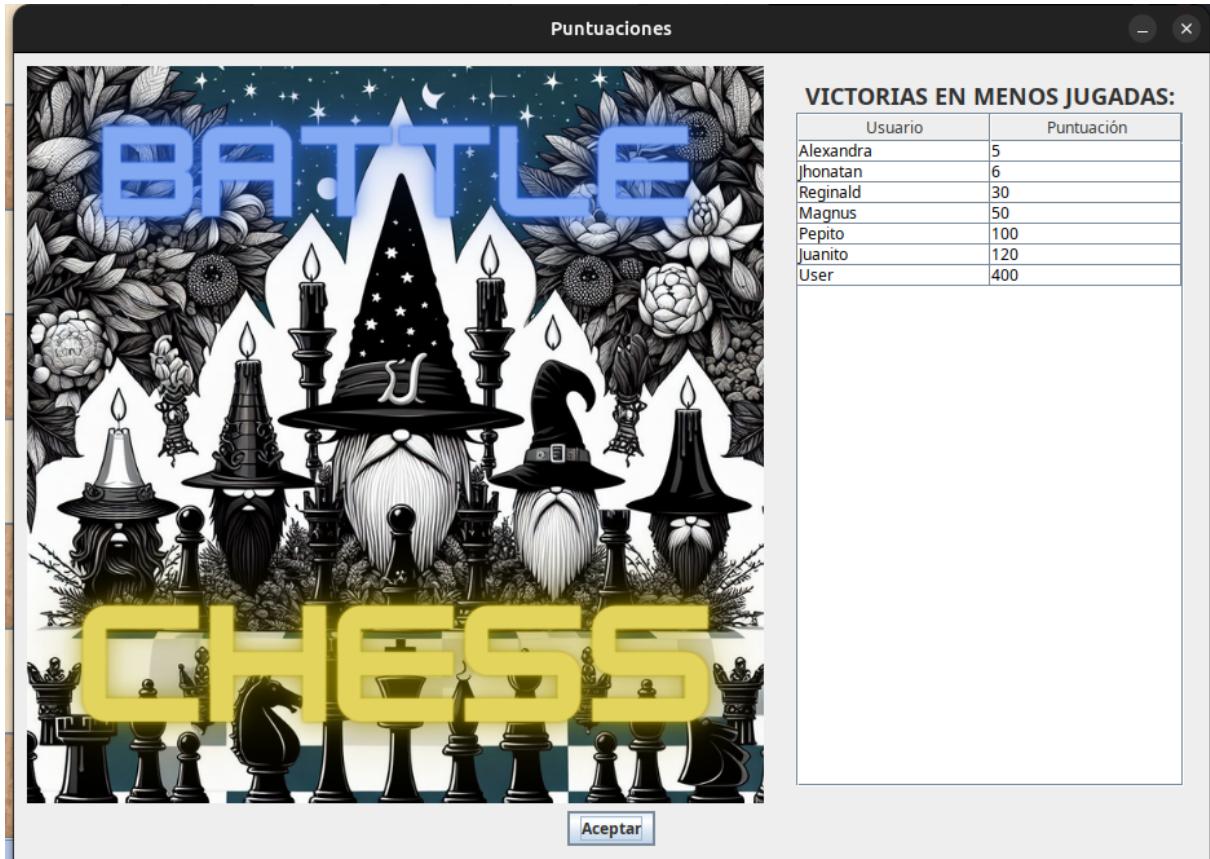
- El juego se desarrolla por turnos, y al realizar un movimiento, las casillas se pintan de rojo o verde según su disponibilidad para el próximo movimiento. Las casillas rojas indican que no están disponibles, mientras que las casillas verdes señalan opciones válidas para realizar el movimiento en el turno actual. Este sistema de visualización facilita al jugador la toma de decisiones estratégicas durante el juego.



- En el ámbito de la base de datos, hemos implementado un sistema de récords que registra las partidas ganadas con el menor número posible de jugadas. Al concluir una partida, se presenta una ventana que permite al jugador guardar su puntuación. Para realizar esta acción, el jugador debe ingresar su nombre y, al hacer clic en "Guardar", la información se registra en la base de datos. Posteriormente, se muestra nuevamente la ventana principal, brindando la opción de iniciar un nuevo juego.



- En caso de que la instancia del juego se corte, aún se puede acceder a los rankings mediante el botón "Ver Ranking". Este botón realiza una consulta a la base de datos, mostrando así las puntuaciones registradas, incluyendo el último registro efectuado, como el jaque mate con 5 jugadas que acabamos de realizar . Esta función garantiza que los récords y las estadísticas de juego estén siempre disponibles para su visualización.



## 8. Rúbricas

### 8.1. 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.

### 8.2. 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

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

Tabla 3: Rúbrica para contenido del Informe y demostración

	Contenido y demostración	Puntos	Checklist	Estudiante	Profesor
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	3	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	1	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	1	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	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	3	
<b>Total</b>		20		19	

## 9. Referencias

- <https://www.geeksforgeeks.org/insertion-sort/>