*Documento de requerimiento*

David Santiago Muñoz Fernández - 20222020054

Santiago Pineda Anaya - 20222020055

**AJEDREZ**

El problema consistía en la creación de un ajedrez funcional en el cual se tenga una pantalla de presentación con un botón de inicio y dos “Labels” donde se pueda escribir el nombre, luego un corto que muestre la historia del juego posterior a eso una pantalla con el ajedrez en el cual tenga un botón para bajar la música y aumentarla otro para cambiarla, un botón que lleve a un video de cómo se juega y otro para cerrar el ajedrez en caso de jaque, y en ultimas imprimir un (.txt) con la cantidad de movimiento con los cuales se terminó la partida ya sea para negras o blancas generando una animación de muerte del rey ya el final de juego después de oprimir ese botón de salida en caso de jaque mate.

1. Análisis conceptual
   1. ¿Qué es un ajedrez?

El ajedrez en uno de los juegos más antiguos, creado por el humano el cual ha dejado a lo largo de la historia problemas celebres y partidas memorables, el ajedrez es un juego de mesa estratégico y competitivo de dos jugadores, cada jugador controla por decirlo así un ejército un rey, una reina, dos alfiles, dos caballos, dos torres y dos peones, en el cual se tiene “ataques llamados jaque y un ataque definitivo denominado jaque mate dados todos al rey siendo la pieza fundamental del juego, conocido como uno de los juegos más completos a nivel estratégico debido a que los individuos buscan prever las jugadas del otro para así ganar la partida en el momento que el rey este en jaque y sin ningún movimiento.



*Gráfica1. Tablero de ajedrez, creación propia.*

* 1. Análisis programación

**Problema:** creación de un ajedrez funcional, con jaques y jaque mate basado en un botón de salida, con interfaces graficas para los nombres y otras opciones en general.

|  |  |
| --- | --- |
| **Cliente** | Docente – estudiantes |
| **Usuario** | Cualquier persona que desee jugar una partida de ajedrez y quiera saber en cuantos movimientos gano una partida |
| **Requerimiento funcional** | 1. Movimiento valido según las reglas del ajedrez de todas las piezas por ejemplo los peones a la salida se pueden mover dos hacia adelante y después solo 1 hacia adelante. 2. Generación de la combinación de piezas entiéndase como enroque, peón al paso, jaque y jaque mate. 3. Generar un archivo txt el cual muestre la cantidad de movimientos validos en los cuales se ganó la partida. |
| **Mundo del problema** | En el enunciado no se explica realmente el mundo del problema el cuna consiste en la lógica de generar el tablero como una matriz 8 x8 en la cual se le agrega la posición de la pieza, además de lógica implementada en las piezas tales como que el rey se puede mover uno hacia arriba menos 1 hacia atrás por ejemplo más adelante se tomara más adelante el código de las piezas, también la validación para que no se salga del tablero además de la implementación de los turnos siendo la lógica necesaria a saber para implementar los booleanos necesarios |
| **Requerimiento no funcional** | La generación de una pantalla de inicio en el cual se tenga los nombres de los jugadores en un label cada uno, la generación de un video de historia del juego, generación de botones para cambiar la música para incrementarla y decrecerla, botón para a un tutorial y cerrar la ventana en caso de jaque, y una animación de muerte de un rey |

|  |  |
| --- | --- |
| **Nombre** | Movimientos de piezas |
| **Resumen** | Dado en evento al mouse generar el movimiento de las piezas |
| **Entradas** | |
| *Evento clic del mouse* | |
| **Resultados** | |
| Movimiento valido de una pieza dentro del tablero | |

|  |  |
| --- | --- |
| **Nombre** | 1. Mostrar resultados |
| **Resumen** | Como salida el programa debe generar dos cosas al final del código la animación e la muerte del rey y un txt con el número de movimientos validos hasta el jaque mate |
| **Entradas** | |
| *Evento del boton x para cerrar la ventana en caso de jaque e* | |
| **Resultados** | |
| Generar un txt el cual diga los movimientos validos hasta el jaque mate, y una animación de la muerte de un rey como salida final | |

**Historia de usuario**

|  |  |  |  |
| --- | --- | --- | --- |
| **Historia de usuario Nro.** | 1 | Titulo: | Eventos de mouse |
| **Descripción** | Como: | Usuario | |
| Quiero: | Presionar la pieza y generar un movimiento valido | |
| Para: | Generar una partida de ajedrez | |
| **Criterios de aceptación** | Las piezas no pueden salir del tablero  Las piezas no pueden generar movimiento errados tal que un rey se muevas más de una casilla, o la reina salte las piezas como un caballo, el caballo no pueda saltar piezas o no se genere un mate bajo las reglas del juego, es decir que cuando sea mate el rey se pueda mover | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Historia de usuario Nro.** | 1 | Titulo: | Calcular el total |
| **Descripción** | Como: | Usuario | |
| Quiero: | Que cuando la x se oprima se genere la salida del programa y al oprimir su botón Genere el link par aun video explicativo de como jugar. | |
| Para: | Terminar el programa | |
| **Criterios de aceptación** | Generar un txt al oprimir ese botón x y la animación de muerte de un rey  Generar un video de explicación para generar un juego limpio a pesar de las restricciones | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Historia de usuario Nro.** | 1 | Titulo: | Introducción al programa |
| **Descripción** | Como: | Usuario | |
| Quiero: | Al iniciar el programa quiero que se genere una interface para la seleccionar del color de piezas, además de un video introductorio al juego | |
| Para: | Tener un contexto del juego | |
| **Criterios de aceptación** | Poder saltar el contexto mediante clicks,ademas poner el nombre en el label de la pieza deseada para seleccionar el color de pieza | | |

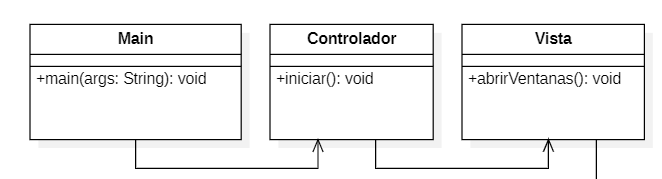
1. Diseño

UML: <https://github.com/Deivis44/Udistrital_Java-Ajedrez/blob/main/INFORME%20%26%20UML/UML_FINAL.pdf>

|  |  |
| --- | --- |
| **Entidad – Clase** | ***Descripción*** |
| Main | Es la clase principal, suministra la iniciación de todos los componentes esenciales del programa a través de la clase “Controlador” que a su vez inicia la clase “Vista”, en este caso hay una interfaz de usuario para que pueda colocar su nombre tanto el jugador 1 como jugador 2, y una vez esto esté terminado inicia otra ventana que realiza una sucesión de imágenes a modo de emular una historia con respecto a el juego, reflejando el comportamiento estándar de un videojuego. |
| Board | Es una clase esencial puesto que es la encargada de utilizar e implementar todas las clases útiles y necesarias para que el juego como tal pueda inicializarse, ya que en realidad este está encima de un frame que tiene otras funcionalidades como música y botones para su control, es por esto que el board adopta y complementa cada herramienta como las clases de las piezas, el movimiento del mouse, los jaques y demás. |

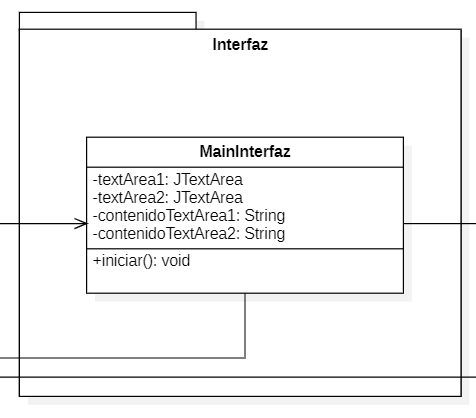
1. *Componentes iniciales*

*(/Chess/src):*

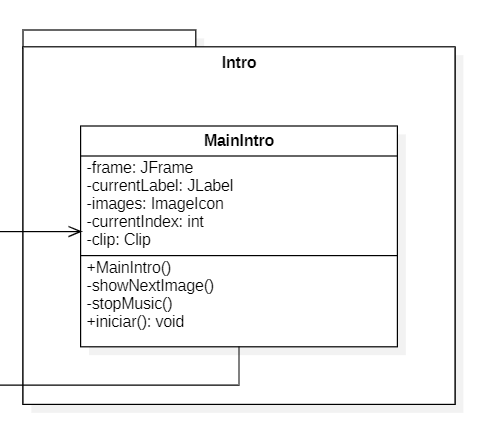


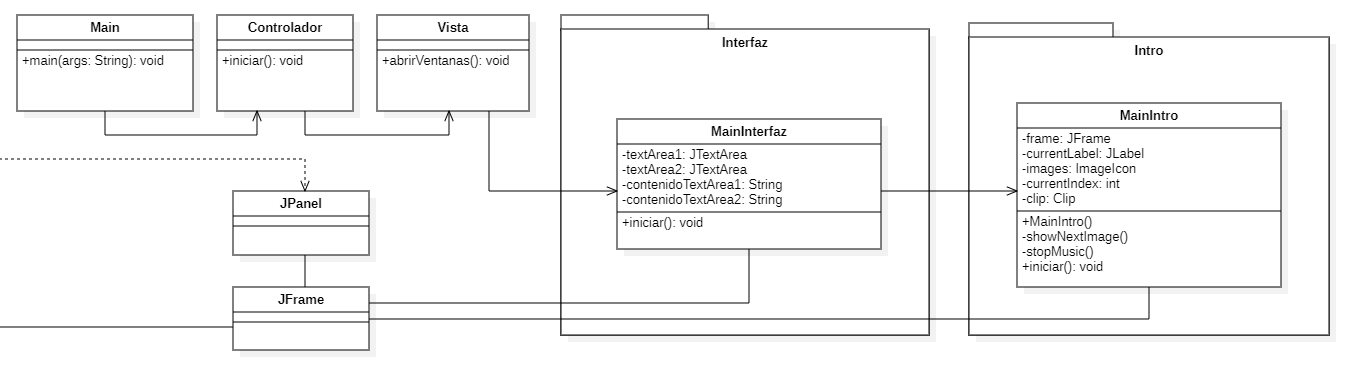
1. Inicio ventanas interfaz de usuario

(*/Chess/src/Interfaz):*



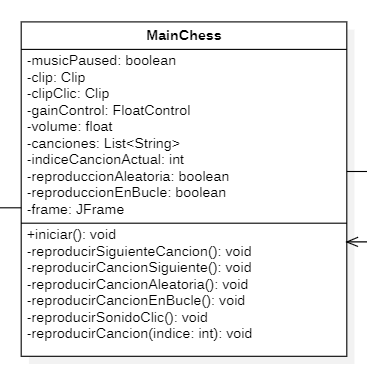
(*/Chess/src/Intro):*



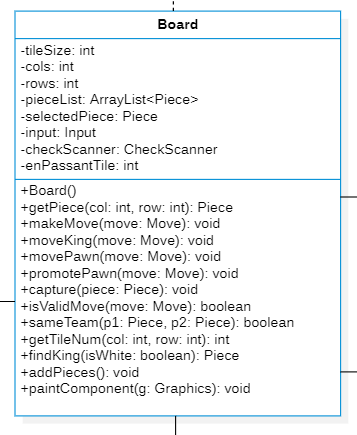


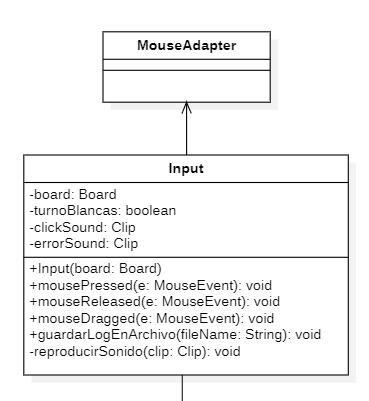
1. Ventana del juego

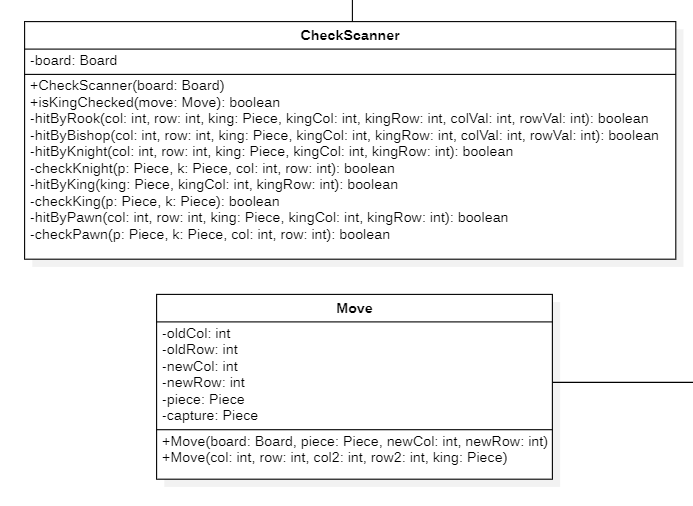
(*/Chess/src/Main):*



1. Llamado tablero del juego

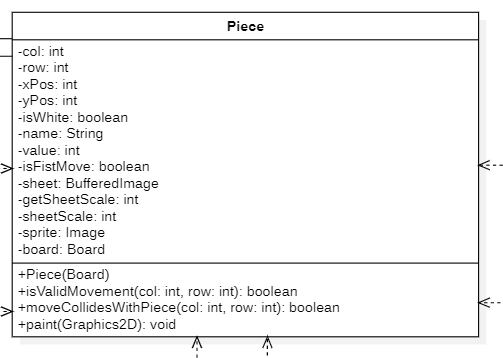


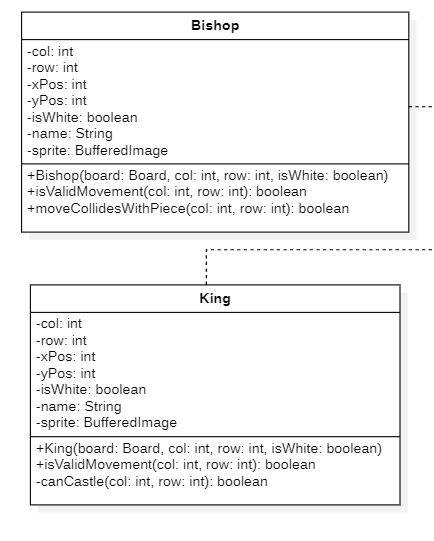


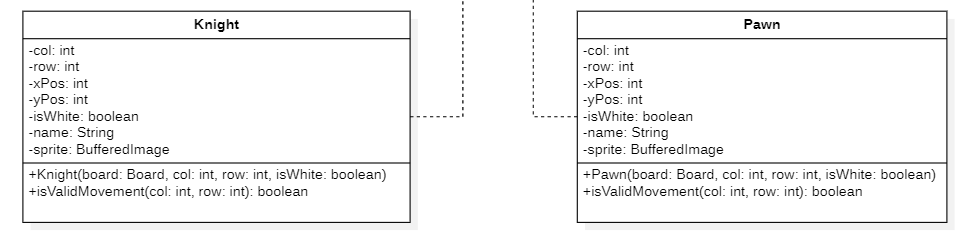


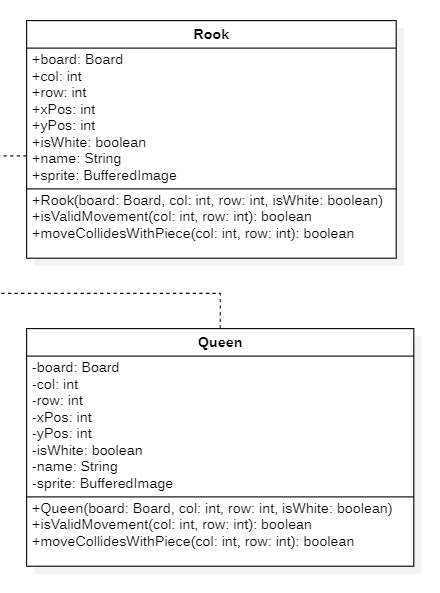
1. Implementación de las piezas

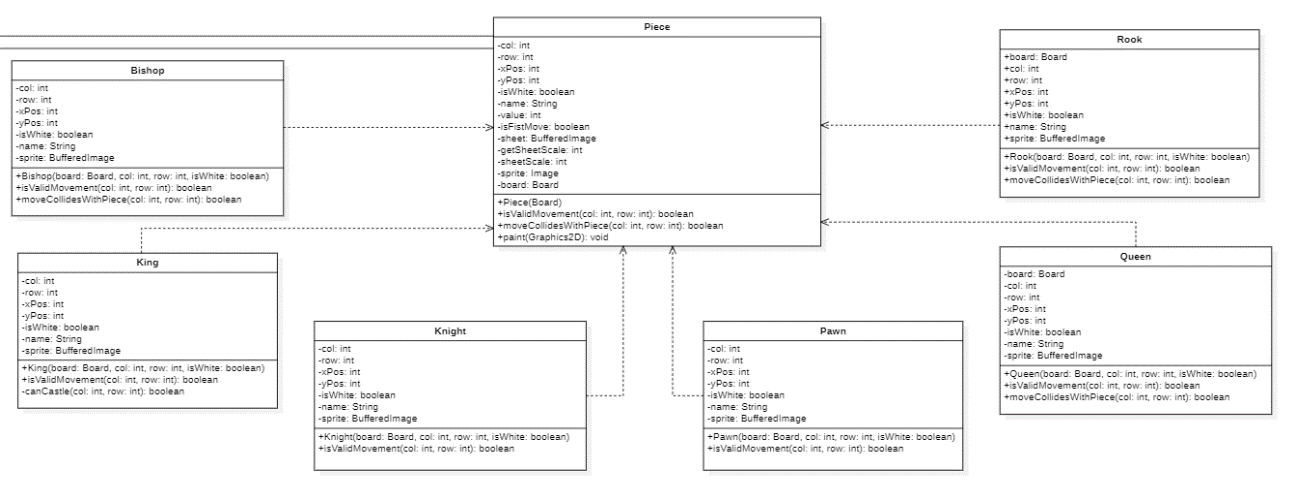
(*/Chess/src/Pieces):*

}

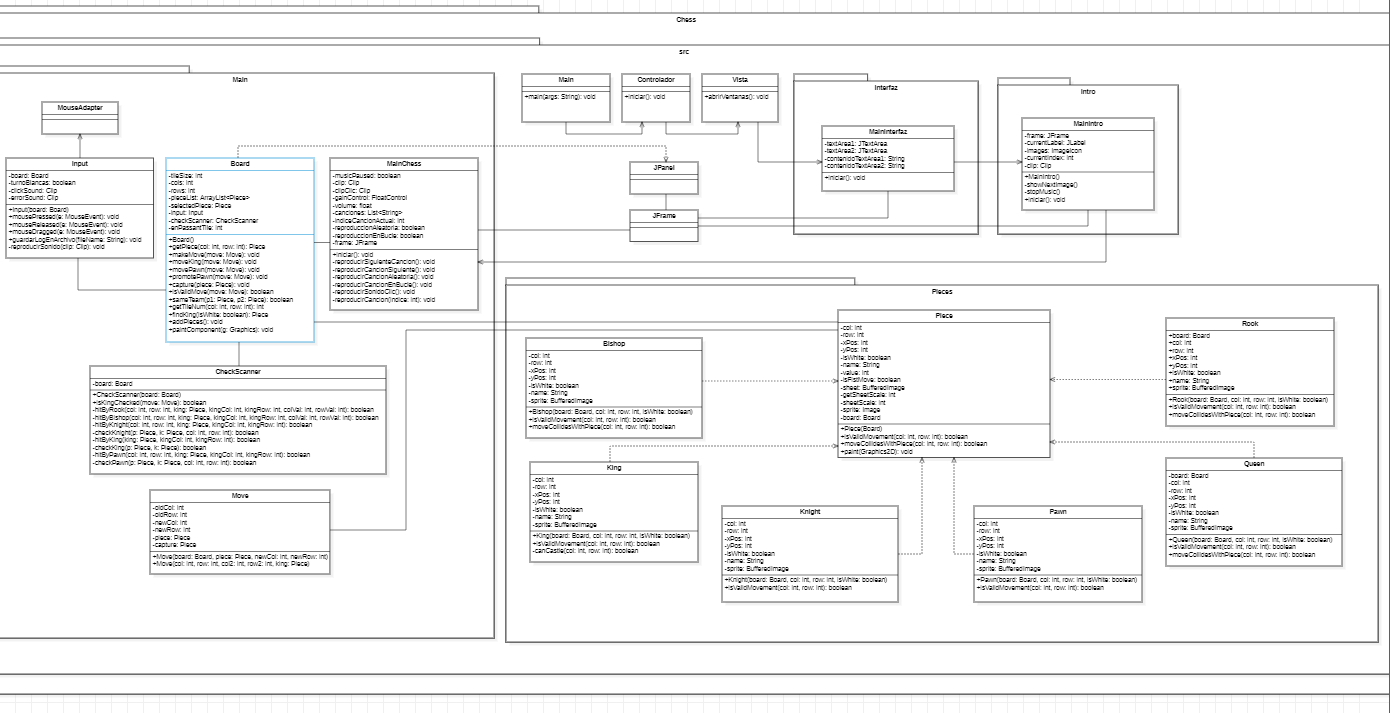








1. UML completo



1. Codificación de la solución

*Link código*: <https://github.com/Deivis44/Udistrital_Java-Ajedrez>

***/Chess***

|  |  |  |  |
| --- | --- | --- | --- |
| *Main.java*   |  | | --- | | public class Main {    public static void main(String[] args) {    Controlador iniciarJuego = new Controlador();  iniciarJuego.iniciar();    }  } |     *Controlador.java*   |  | | --- | | public class Controlador {    public void iniciar() {  Vista iniciarJuego = new Vista();  iniciarJuego.abrirVentanas();  }  } |     *Vista.java*   |  | | --- | | *import Interfaz.MainInterfaz;*    *public class Vista {*    *public void abrirVentanas() {*  *MainInterfaz intro = new MainInterfaz();*  *intro.iniciar();*  *}*  *}* | |

***/Interfaz***

|  |  |
| --- | --- |
| *MainInterfaz.java*   |  | | --- | | package Interfaz;    import javax.swing.\*;    import Intro.MainIntro;    import java.awt.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.awt.image.BufferedImage;    import javax.imageio.ImageIO;  import javax.sound.sampled.\*;    import java.io.File;  import java.io.IOException;    public class MainInterfaz {    public JTextArea textArea1;  public JTextArea textArea2;  public static String contenidoTextArea1;  public static String contenidoTextArea2;    public JTextArea textArea;    public void iniciar() {    // ! musica  final Clip[] clip = new Clip[1]; // Declarar un arreglo final de Clip para acceder a él desde el ActionListener  try {  // música  AudioInputStream inputStream = AudioSystem.getAudioInputStream(MainInterfaz.class.getResourceAsStream("/musica/theme1.wav"));  clip[0] = AudioSystem.getClip();  clip[0].open(inputStream);  clip[0].loop(Clip.LOOP\_CONTINUOUSLY); // Loop  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }    // ? JFrame  ImageIcon icono = new ImageIcon("Chess/src/Interfaz/Imagenes/icon.png");  final String title = "| CHESS | Elije a tu personaje";    JFrame frame = new JFrame();  frame.setIconImage(icono.getImage());  frame.setTitle(title);  frame.getContentPane().setBackground(Color.BLACK);  frame.setLayout(new GridBagLayout());  frame.setMinimumSize(new Dimension(1200,715));  frame.setLocationRelativeTo(null);    frame.setResizable(false);  frame.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);    ImageIcon fondo = new ImageIcon("Chess/src/Interfaz/Imagenes/dragon\_resize2.gif");  JLabel imagen = new JLabel(fondo);  imagen.setVisible(true);  frame.add(imagen);    frame.setVisible(true);      // titulo  ImageIcon titulo = new ImageIcon("Chess/src/Interfaz/Addons/title.gif");  JLabel imgtitulo1 = new JLabel(titulo);  imgtitulo1.setBounds(0, 48, 470, 185);  imgtitulo1.setVisible(true);  imagen.add(imgtitulo1);    // puntero rey1  ImageIcon flecha1 = new ImageIcon("Chess/src/Interfaz/Addons/arrow.gif");  JLabel imgflecha1 = new JLabel(flecha1);  imgflecha1.setBounds(725, 340, 50, 50);  imgflecha1.setVisible(true);  imagen.add(imgflecha1);    // puntero rey2  ImageIcon flecha2 = new ImageIcon("Chess/src/Interfaz/Addons/arrow.gif");  JLabel imgflecha2 = new JLabel(flecha2);  imgflecha2.setBounds(895, 340, 50, 50);  imgflecha2.setVisible(true);  imagen.add(imgflecha2);    // rey blanco  ImageIcon rey1 = new ImageIcon("Chess/src/Interfaz/Sprites/Rey Blanco/idle.gif");  JLabel imgrey1 = new JLabel(rey1);  imgrey1.setBounds(680, 400, 120, 120);  imgrey1.setVisible(true);  imagen.add(imgrey1);    // rey negro  ImageIcon rey2 = new ImageIcon("Chess/src/Interfaz/Sprites/Rey Negro/idle.gif");  JButton imgrey2 = new JButton(rey2);  imgrey2.setBounds(875, 400, 120, 120);  imgrey2.setOpaque(false);  imgrey2.setContentAreaFilled(false);  imgrey2.setBorderPainted(false);  imgrey2.setVisible(true);  imagen.add(imgrey2);      // botones  ImageIcon imgPlay = new ImageIcon("imagenes/Addons/play.png");  JButton btnPlay = new JButton(imgPlay);  btnPlay.setBounds(36, 563, 150, 90);  btnPlay.setOpaque(false);  btnPlay.setContentAreaFilled(false);  btnPlay.setBorderPainted(false);  btnPlay.setVisible(true);  imagen.add(btnPlay);    btnPlay.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  if (clip[0] != null && clip[0].isRunning()) {  clip[0].stop(); // Detiene la reproducción del audio  }    frame.dispose(); // Cierra la ventana actual    // Obtener el contenido de los JTextAreas  contenidoTextArea1 = textArea1.getText();  contenidoTextArea2 = textArea2.getText();    // Código para abrir la nueva ventana  MainIntro intro = new MainIntro();  intro.iniciar();  }  });      // ImageIcon imgP1 = new ImageIcon("imagenes/Addons/player1.png");  // JLabel P1 = new JLabel(imgP1);  // P1.setBounds(196, 608, 105, 45);  // P1.setVisible(true);  // imagen.add(P1);    // ImageIcon imgP2 = new ImageIcon("imagenes/Addons/player2.png");  // JLabel P2 = new JLabel(imgP2);  // P2.setBounds(309, 608, 105, 45);  // P2.setVisible(true);  // imagen.add(P2);    // Text Area personalizado con imagen de fondo  textArea1 = new CustomTextArea("imagenes/Addons/player1.png");  textArea1.setBounds(196, 608, 105, 45);  imagen.add(textArea1);    textArea2 = new CustomTextArea("imagenes/Addons/player2.png");  textArea2.setBounds(309, 608, 105, 45);  imagen.add(textArea2);  }    private static class CustomTextArea extends JTextArea {    private BufferedImage backgroundImage;    public CustomTextArea(String imageFilePath) {  // Cargar la imagen de fondo  try {  backgroundImage = ImageIO.read(new File(imageFilePath));  } catch (IOException e) {  e.printStackTrace();  }  setOpaque(false); // Hacer que el JTextArea sea transparente  setFont(new Font("Impact", Font.PLAIN, 30)); // Establecer la fuente del texto    }    @Override  protected void paintComponent(Graphics g) {  // Dibujar la imagen de fondo  if (backgroundImage != null) {  g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);  }  super.paintComponent(g);  }  }  } | |

***/Intro***

|  |  |
| --- | --- |
| *MainIntro.java*   |  | | --- | | package Intro;    import javax.swing.\*;    import Main.MainChess;    import java.awt.\*;  import javax.sound.sampled.\*;  import java.io.IOException;  import java.awt.event.\*;    public class MainIntro {    private JFrame frame;  private JLabel currentLabel;  private ImageIcon[] images = {  new ImageIcon("Chess/src/Intro/Imagenes/1.gif"),  new ImageIcon("Chess/src/Intro/Imagenes/2.gif"),  new ImageIcon("Chess/src/Intro/Imagenes/3.gif"),  new ImageIcon("Chess/src/Intro/Imagenes/4.gif"),  new ImageIcon("Chess/src/Intro/Imagenes/5.gif")  };  private int currentIndex = 0;    private Clip clip;    public MainIntro() {  ImageIcon icono = new ImageIcon("imagenes/Recursos/icono3.png");  final String title = "| CHESS | Intro";    frame = new JFrame();  frame.setTitle(title);  frame.setIconImage(icono.getImage());  frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  frame.getContentPane().setBackground(Color.BLACK);  frame.setSize(1200, 940);  frame.setLayout(new BorderLayout());  frame.setLocationRelativeTo(null);  frame.setResizable(false);    currentLabel = new JLabel();  currentLabel.setHorizontalAlignment(SwingConstants.CENTER);  currentLabel.setVerticalAlignment(SwingConstants.CENTER);  frame.add(currentLabel, BorderLayout.CENTER);    frame.addMouseListener(new java.awt.event.MouseAdapter() {  public void mouseClicked(java.awt.event.MouseEvent evt) {  showNextImage();  }  });    showNextImage();    frame.addWindowListener(new WindowAdapter() {  public void windowClosed(WindowEvent e) {  SwingUtilities.invokeLater(new Runnable() {  public void run() {  stopMusic();  MainChess main = new MainChess();  main.iniciar();  }  });  }  });    frame.setVisible(true);    // Reproducir la canción de fondo  try {  AudioInputStream inputStream = AudioSystem.getAudioInputStream(MainIntro.class.getResourceAsStream("/musica/title.wav"));  clip = AudioSystem.getClip();  clip.open(inputStream);  clip.loop(Clip.LOOP\_CONTINUOUSLY); // Loop  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }  }    private void showNextImage() {  if (currentIndex < images.length) {  currentLabel.setIcon(images[currentIndex]);  currentIndex++;  } else {  frame.dispose();  }  }    private void stopMusic() {  if (clip != null) {  clip.stop();  clip.close();  }  }      public void iniciar() {  SwingUtilities.invokeLater(new Runnable() {  public void run() {  }  });  }  } | |

***/Main***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Board.java*   |  | | --- | | package Main;    import Pieces.\*;    import javax.swing.\*;  import java.awt.\*;  import java.util.ArrayList;    public class Board extends JPanel {    public int tileSize = 85;  public int cols = 8;  int rows = 8;    ArrayList<Piece> pieceList = new ArrayList<>();    public Piece selectedPiece;    Input input = new Input(this);    public CheckScanner checkScanner = new CheckScanner(this);    public int enPassantTile = -1;    public Board() {  this.setPreferredSize(new Dimension(cols \* tileSize, rows \* tileSize));    this.addMouseListener(input);  this.addMouseMotionListener(input);    addPieces();  }    public Piece getPiece(int col, int row) {    for (Piece piece: pieceList) {  if (piece.col == col && piece.row == row) {  return piece;  }  }    return null;  }    public void makeMove(Move move) {    if (move.piece.name.equals("Pawn")) {  movePawn(move);  } else if (move.piece.name.equals("King")) {  moveKing((move));  }    move.piece.col = move.newCol;  move.piece.row = move.newRow;  move.piece.xPos = move.newCol \* tileSize;  move.piece.yPos = move.newRow \* tileSize;    move.piece.isFistMove = false;    capture(move.capture);  }    private void moveKing(Move move) {    if (Math.abs(move.piece.col - move.newCol) == 2) {  Piece rook;  if (move.piece.col < move.newCol) {  rook = getPiece(7, move.piece.row);  rook.col = 5;  } else {  rook = getPiece(0, move.piece.row);  rook.col = 3;  }    rook.xPos = rook.col \* tileSize;  }  }    private void movePawn(Move move) {    // en passant  int colorIndex = move.piece.isWhite ? 1 : -1;    if (getTileNum(move.newCol, move.newRow) == enPassantTile) {  move.capture = getPiece(move.newCol, move.newRow + colorIndex);  }  if (Math.abs(move.piece.row - move.newRow) == 2) {  enPassantTile = getTileNum(move.newCol, move.newRow + colorIndex);  } else {  enPassantTile = -1;  }    // promotions  colorIndex = move.piece.isWhite ? 0 : 7;  if (move.newRow == colorIndex) {  promotePawn(move);  }  }    private void promotePawn(Move move) {  pieceList.add(new Queen(this, move.newCol, move.newRow, move.piece.isWhite));  capture(move.piece);  }    public void capture(Piece piece) {  pieceList.remove(piece);  }    public boolean isValidMove(Move move) {    if (sameTeam(move.piece, move.capture)) {  return false;  }  if (!move.piece.isValidMovement(move.newCol, move.newRow)) {  return false;  }  if (move.piece.moveCollidesWithPiece(move.newCol, move.newRow)) {  return false;  }    if (checkScanner.isKingChecked(move)) {  return false;  }      return true;  }    public boolean sameTeam(Piece p1, Piece p2) {  if (p1 == null || p2 == null) {  return false;  }  return p1.isWhite == p2.isWhite;  }    public int getTileNum(int col, int row) {  return row \* rows + col;  }    Piece findKing(boolean isWhite) {  for (Piece piece : pieceList) {  if (isWhite == piece.isWhite && piece.name.equals("King")) {  return piece;  }  }  return null;  }    public void addPieces() {  pieceList.add(new Rook(this, 0, 0, false));  pieceList.add(new Knight(this, 1, 0, false));  pieceList.add(new Bishop(this, 2, 0, false));  pieceList.add(new Queen(this, 3, 0, false));  pieceList.add(new King(this, 4, 0, false));  pieceList.add(new Bishop(this, 5, 0, false));  pieceList.add(new Knight(this, 6, 0, false));  pieceList.add(new Rook(this, 7, 0, false));    pieceList.add(new Pawn(this, 0, 1, false));  pieceList.add(new Pawn(this, 1, 1, false));  pieceList.add(new Pawn(this, 2, 1, false));  pieceList.add(new Pawn(this, 3, 1, false));  pieceList.add(new Pawn(this, 4, 1, false));  pieceList.add(new Pawn(this, 5, 1, false));  pieceList.add(new Pawn(this, 6, 1, false));  pieceList.add(new Pawn(this, 7, 1, false));    pieceList.add(new Rook(this, 0, 7, true));  pieceList.add(new Knight(this, 1, 7, true));  pieceList.add(new Bishop(this, 2, 7, true));  pieceList.add(new Queen(this, 3, 7, true));  pieceList.add(new King(this, 4, 7, true));  pieceList.add(new Bishop(this, 5, 7, true));  pieceList.add(new Knight(this, 6, 7, true));  pieceList.add(new Rook(this, 7, 7, true));    pieceList.add(new Pawn(this, 0, 6, true));  pieceList.add(new Pawn(this, 1, 6, true));  pieceList.add(new Pawn(this, 2, 6, true));  pieceList.add(new Pawn(this, 3, 6, true));  pieceList.add(new Pawn(this, 4, 6, true));  pieceList.add(new Pawn(this, 5, 6, true));  pieceList.add(new Pawn(this, 6, 6, true));  pieceList.add(new Pawn(this, 7, 6, true));    }    public void paintComponent(Graphics g) {  Graphics2D g2d = (Graphics2D) g;    // paint board  for (int r = 0; r < rows; r++) {  for (int c = 0; c < cols; c++) {  g2d.setColor((c + r) % 2 == 0 ? new Color(166, 186, 196) : new Color(77, 34, 51));  g2d.fillRect(c \* tileSize, r \* tileSize, tileSize , tileSize);  }  }    // paint highlights  if (selectedPiece != null)  for (int r = 0; r < rows; r++)  for (int c = 0; c < cols; c++) {  if (isValidMove(new Move(this, selectedPiece, c, r))) {  g2d.setColor(new Color(1, 1, 1, 128));  g2d.fillRect(c \* tileSize, r \* tileSize, tileSize, tileSize);  }  }    // paint pieces  for (Piece piece : pieceList) {  piece.paint(g2d);  }  }  } |     *CheckScanner.java*   |  | | --- | | package Main;    import Pieces.Piece;    public class CheckScanner {    Board board;    public CheckScanner(Board board) {  this.board = board;  }    public boolean isKingChecked(Move move) {  Piece king = board.findKing(move.piece.isWhite);  assert king != null;    int kingCol = king.col;  int kingRow = king.row;    if (board.selectedPiece != null && board.selectedPiece.name.equals("King")) {  kingCol = move.newCol;  kingRow = move.newRow;  }    boolean isKingChecked = hitByRook(move.newCol, move.newRow, king, kingCol, kingRow, 0, 1) ||  hitByRook(move.newCol, move.newRow, king, kingCol, kingRow, 1, 0) ||  hitByRook(move.newCol, move.newRow, king, kingCol, kingRow, 0, -1) ||  hitByRook(move.newCol, move.newRow, king, kingCol, kingRow, -1, 0) ||  hitByBishop(move.newCol, move.newRow, king, kingCol, kingRow, -1, -1) ||  hitByBishop(move.newCol, move.newRow, king, kingCol, kingRow, 1, -1) ||  hitByBishop(move.newCol, move.newRow, king, kingCol, kingRow, 1, 1) ||  hitByBishop(move.newCol, move.newRow, king, kingCol, kingRow, -1, 1) ||  hitByKnight(move.newCol, move.newRow, king, kingCol, kingRow) ||  hitByPawn(move.newCol, move.newRow, king, kingCol, kingRow) ||  hitByKing(king, kingCol, kingRow);    return isKingChecked;  }      private boolean hitByRook(int col, int row, Piece king, int kingCol, int kingRow, int colVal, int rowVal) {    for (int i = 1; i < 8; i++) {  if (kingCol + (i \* colVal) == col && kingRow + (i \* rowVal) == row) {  break;  }    Piece piece = board.getPiece(kingCol + (i \* colVal), kingRow + (i \* rowVal));  if (piece != null && piece != board.selectedPiece) {  if (!board.sameTeam(piece, king) && (piece.name.equals("Rook") || piece.name.equals("Queen"))) {  return true;  } break;  }  }    return false;  }    private boolean hitByBishop(int col, int row, Piece king, int kingCol, int kingRow, int colVal, int rowVal) {    for (int i = 1; i < 8; i++) {  if (kingCol - (i \* colVal) == col && kingRow - (i \* rowVal) == row) {  break;  }    Piece piece = board.getPiece(kingCol - (i \* colVal), kingRow - (i \* rowVal));  if (piece != null && piece != board.selectedPiece) {  if (!board.sameTeam(piece, king) && (piece.name.equals("Bishop") || piece.name.equals("Queen"))) {  return true;  } break;  }  }    return false;  }    private boolean hitByKnight(int col, int row, Piece king, int kingCol, int kingRow) {  return checkKnight(board.getPiece(kingCol - 1, kingRow - 2), king, col, row) ||  checkKnight(board.getPiece(kingCol + 1, kingRow - 2), king, col, row) ||  checkKnight(board.getPiece(kingCol + 2, kingRow - 1), king, col, row) ||  checkKnight(board.getPiece(kingCol + 2, kingRow + 1), king, col, row) ||  checkKnight(board.getPiece(kingCol + 1, kingRow + 2), king, col, row) ||  checkKnight(board.getPiece(kingCol - 1, kingRow + 2), king, col, row) ||  checkKnight(board.getPiece(kingCol - 2, kingRow + 1), king, col, row) ||  checkKnight(board.getPiece(kingCol - 2, kingRow - 1), king, col, row);  }    private boolean checkKnight(Piece p, Piece k, int col, int row) {  return p != null && !board.sameTeam(p, k) && p.name.equals("Knight") && !(p.col == col && p.row == row);  }    private boolean hitByKing(Piece king, int kingCol, int kingRow) {  return checkKing(board.getPiece(kingCol - 1, kingRow - 1), king) ||  checkKing(board.getPiece(kingCol + 1, kingRow - 1), king) ||  checkKing(board.getPiece(kingCol,kingRow - 1), king) ||  checkKing(board.getPiece(kingCol - 1, kingRow), king) ||  checkKing(board.getPiece(kingCol + 1, kingRow), king) ||  checkKing(board.getPiece(kingCol - 1, kingRow + 1), king) ||  checkKing(board.getPiece(kingCol + 1, kingRow + 1), king) ||  checkKing(board.getPiece(kingCol, kingRow + 1), king);  }    private boolean checkKing(Piece p, Piece k) {  return p != null && !board.sameTeam(p, k) && p.name.equals("King");  }    private boolean hitByPawn(int col, int row, Piece king, int kingCol, int kingRow) {  int colorVal = king.isWhite ? -1 : 1;    return checkPawn(board.getPiece(kingCol + 1, kingRow + colorVal), king, col, row) ||  checkPawn(board.getPiece(kingCol - 1, kingRow + colorVal), king, col, row);  }    private boolean checkPawn(Piece p, Piece k, int col, int row) {  return p != null && !board.sameTeam(p, k) && p.name.equals("Pawn") && !(p.col == col && p.row == row);  }    // programar el maldito mate me hizo perder 40 años de vida 30/05/23 - 3:57AM  } |     *Input.java*   |  | | --- | | package Main;    import Pieces.Piece;    import javax.sound.sampled.\*;  import java.awt.event.MouseAdapter;  import java.awt.event.MouseEvent;  import java.io.IOException;  import java.io.BufferedWriter;  import java.io.FileWriter;    public class Input extends MouseAdapter {    private Board board;  private boolean turnoBlancas = true;  public static int contador = 1;  public static StringBuilder logBuilder = new StringBuilder();  Clip clickSound; // Clip para el sonido de clic;  Clip errorSound; // Clip para el sonido de error    public Input(Board board) {  this.board = board;    try {  // Obtener el archivo de audio del sonido de clic  AudioInputStream clickInputStream = AudioSystem.getAudioInputStream(Input.class.getResourceAsStream("/musica/efectos/click.wav"));  // Cargar el clip de audio para el sonido de clic  clickSound = AudioSystem.getClip();  clickSound.open(clickInputStream);    // Obtener el archivo de audio del sonido de error  AudioInputStream errorInputStream = AudioSystem.getAudioInputStream(Input.class.getResourceAsStream("/musica/efectos/error.wav"));  // Cargar el clip de audio para el sonido de error  errorSound = AudioSystem.getClip();  errorSound.open(errorInputStream);  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }  }    @Override  public void mousePressed(MouseEvent e) {  int col = e.getX() / board.tileSize;  int row = e.getY() / board.tileSize;    Piece pieceXY = board.getPiece(col, row);    if (pieceXY != null) {  board.selectedPiece = pieceXY;  reproducirSonido(clickSound);  }  }        public static String movimiento;  @Override  public void mouseReleased(MouseEvent e) {  int col = e.getX() / board.tileSize;  int row = e.getY() / board.tileSize;    if (board.selectedPiece != null) {  Move move = new Move(board, board.selectedPiece, col, row);    if (board.isValidMove(move) && ((turnoBlancas && board.selectedPiece.isWhite) || (!turnoBlancas && !board.selectedPiece.isWhite))) {  board.makeMove(move);    if (contador < 10) {  movimiento = String.format("[%d ] Movimiento ", contador);  } else {  movimiento = String.format("[%d] Movimiento ", contador);  }    if (contador % 2 == 0) {  movimiento += "negras: válido |";  } else {  movimiento += "blancas: válido |";  }    System.out.println(movimiento);  logBuilder.append(movimiento).append(System.lineSeparator());  contador++;    turnoBlancas = !turnoBlancas;  reproducirSonido(clickSound);  guardarLogEnArchivo("Chess\_moves.txt"); // Guardar el registro en un archivo de texto    } else {  board.selectedPiece.xPos = board.selectedPiece.col \* board.tileSize;  board.selectedPiece.yPos = board.selectedPiece.row \* board.tileSize;  reproducirSonido(errorSound);  }  }    board.selectedPiece = null;  board.repaint();  }    private void reproducirSonido(Clip clip) {  if (clip != null) {  clip.stop(); // Detener el sonido si se está reproduciendo  clip.setFramePosition(0); // Reiniciar la reproducción desde el inicio  clip.start(); // Reproducir el sonido  }  }      @Override  public void mouseDragged(MouseEvent e) {    if (board.selectedPiece != null) {  board.selectedPiece.xPos = e.getX() - board.tileSize / 2;  board.selectedPiece.yPos = e.getY() - board.tileSize / 2;    board.repaint();  }  }    public static void guardarLogEnArchivo(String fileName) {  String currentDir = System.getProperty("user.dir");  String filePath = currentDir + "/" + fileName;    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {  writer.write("\nCHESS JAVA - UD ------------------------ |\n |\n"); // Línea de título    writer.write(logBuilder.toString()); // Líneas de movimiento  System.out.println("\n\_archivo .txt modificado\_\n");  } catch (IOException e) {  System.out.println("\n\_error en la modificación del .txt\_: \n" + e.getMessage());  }  }  } |     *MainChess.java*   |  | | --- | | package Main;    import Interfaz.MainInterfaz;    import java.awt.\*;  import javax.swing.\*;  import java.awt.event.\*;  import javax.sound.sampled.\*;  import java.io.IOException;    import java.util.Arrays;  import java.util.List;  import java.util.Random;    import java.awt.Desktop;  import java.net.URI;  import java.net.URISyntaxException;    public class MainChess {    private boolean musicPaused = false;  private Clip clip; // musica en general  private Clip clipClic; // Sonido de clic  private FloatControl gainControl;  private float volume = 0.0f; // Nivel de volumen inicial = 100%  private List<String> canciones = Arrays.asList("battle1.wav", "battle2.wav", "icy.wav", "theme2.wav");  private int indiceCancionActual = 0;  private boolean reproduccionAleatoria = false;  private boolean reproduccionEnBucle = false;    public static JFrame frame = new JFrame();  public void iniciar() {    // titulo por consola  final String copyRight = "Universidad Distrital - Ajedrez | © 2023";    ImageIcon icono = new ImageIcon("imagenes/Recursos/icono.png");    frame.setIconImage(icono.getImage());  frame.setTitle(copyRight);  frame.getContentPane().setBackground(Color.BLACK);  frame.setLayout(new GridBagLayout());  frame.setMinimumSize(new Dimension(1400,940));  frame.setLocationRelativeTo(null);    frame.setResizable(false);  frame.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);    ImageIcon fondo = new ImageIcon("imagenes/Previews/layout-juego7.png");  JLabel imagen = new JLabel(fondo);  imagen.setVisible(true);  frame.add(imagen);    try {  // Obtener el archivo de audio del sonido de clic  AudioInputStream clickInputStream = AudioSystem.getAudioInputStream(Input.class.getResourceAsStream("/musica/efectos/mouse.wav"));  // Cargar el clip de audio para el sonido de clic  clipClic = AudioSystem.getClip();  clipClic.open(clickInputStream);  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }    // ? seccion botones control  ImageIcon imgBotonMusica = new ImageIcon("imagenes/Addons/musica.png");  JButton botonMusica = new JButton(imgBotonMusica);  botonMusica.setBounds(1220, 60, 50, 55);  botonMusica.setOpaque(false);  botonMusica.setContentAreaFilled(false);  botonMusica.setBorderPainted(false);  imagen.add(botonMusica);    botonMusica.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  reproducirSonidoClic();  }  });    ImageIcon imgBotonClose = new ImageIcon("imagenes/Addons/close.png");  JButton botonClose = new JButton(imgBotonClose);  botonClose.setBounds(1279, 60, 50, 55);  botonClose.setOpaque(false);  botonClose.setContentAreaFilled(false);  botonClose.setBorderPainted(false);  imagen.add(botonClose);  botonClose.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  frame.dispose(); // Cierra la ventana actual    String jaqueMate;    if (Input.contador % 2 == 0) {  jaqueMate = "\n |\n ¡Jaque mate de blancas! |\n";  } else {  jaqueMate = "\n |\n ¡Jaque mate de negras! |\n";  }    Input.logBuilder.append(jaqueMate).append(System.lineSeparator());  Input.guardarLogEnArchivo("Chess\_moves.txt");  }  });      ImageIcon imgBotonInfo = new ImageIcon("imagenes/Addons/info.png");  JButton botonInfo = new JButton(imgBotonInfo);  botonInfo.setBounds(1215, 130, 120, 40);  botonInfo.setOpaque(false);  botonInfo.setContentAreaFilled(false);  botonInfo.setBorderPainted(false);  imagen.add(botonInfo);  botonInfo.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  String videoURL = "<https://www.youtube.com/watch?v=AH4Vx5zz7Go>";  Desktop.getDesktop().browse(new URI(videoURL));  } catch (IOException | URISyntaxException ex) {  ex.printStackTrace();  }  }  });      // ? seccion música  ImageIcon imgBotonPausa = new ImageIcon("imagenes/Addons/pausa.png");  JButton botonPausa = new JButton(imgBotonPausa);  botonPausa.setBounds(1090, 65, 50, 50);  botonPausa.setOpaque(false);  botonPausa.setContentAreaFilled(false);  botonPausa.setBorderPainted(false);  imagen.add(botonPausa);    botonPausa.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  reproducirSonidoClic();  }  });    ImageIcon imgBotonMas = new ImageIcon("imagenes/Addons/mas.png");  JButton botonMas = new JButton(imgBotonMas);  botonMas.setBounds(1140, 65, 50, 50);  botonMas.setOpaque(false);  botonMas.setContentAreaFilled(false);  botonMas.setBorderPainted(false);  imagen.add(botonMas);    botonMas.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  reproducirSonidoClic();  }  });    ImageIcon imgBotonMenos = new ImageIcon("imagenes/Addons/menos.png");  JButton botonMenos = new JButton(imgBotonMenos);  botonMenos.setBounds(1040, 65, 50, 50);  botonMenos.setOpaque(false);  botonMenos.setContentAreaFilled(false);  botonMenos.setBorderPainted(false);  imagen.add(botonMenos);    botonMenos.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  reproducirSonidoClic();  }  });      // ? texareas  JLabel player1 = new TransparentLabel(MainInterfaz.contenidoTextArea1);  player1.setFont(new Font("Impact", Font.PLAIN, 24));  player1.setHorizontalAlignment(SwingConstants.CENTER);  player1.setBounds(940, 582, 90, 40);  imagen.add(player1);    JLabel player2 = new TransparentLabel(MainInterfaz.contenidoTextArea2);  player2.setFont(new Font("Impact", Font.PLAIN, 24));  player2.setHorizontalAlignment(SwingConstants.CENTER);  player2.setBounds(1154, 582, 90, 40);  imagen.add(player2);      // ! inicializo tablero con todo el codigo del ajedrez  Board board = new Board();  board.setBounds(126, 85, 680, 680);  imagen.add(board);    frame.setVisible(true);      try {  // Obtener el archivo de audio  AudioInputStream inputStream = AudioSystem.getAudioInputStream(MainChess.class.getResourceAsStream("/musica/" + canciones.get(indiceCancionActual)));    // Cargar el clip de audio  clip = AudioSystem.getClip();  clip.open(inputStream);    // Obtener el control de volumen del clip  gainControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER\_GAIN);    // Ajustar el volumen inicial  gainControl.setValue(volume);    // Reproducir en bucle continuo  clip.loop(Clip.LOOP\_CONTINUOUSLY);  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }    botonPausa.addActionListener(e -> {  if (musicPaused) {  clip.start(); // Reanudar la reproducción de la música  musicPaused = false;  } else {  clip.stop(); // Pausar la reproducción de la música  musicPaused = true;  }  });    botonMas.addActionListener(e -> {  if (volume < 0.0f) {  volume += 5.0f; // Incrementar el volumen en 5 decibelios  gainControl.setValue(volume); // Aplicar el nuevo volumen al control  }  });    botonMenos.addActionListener(e -> {  if (volume > -80.0f) {  volume -= 5.0f; // Decrementar el volumen en 5 decibelios  gainControl.setValue(volume); // Aplicar el nuevo volumen al control  }  });    botonMusica.addActionListener(e -> {  reproducirSiguienteCancion();  });  }    private void reproducirSiguienteCancion() {  if (reproduccionAleatoria) {  reproducirCancionAleatoria();  } else if (reproduccionEnBucle) {  reproducirCancionEnBucle();  } else {  reproducirCancionSiguiente();  }  }    private void reproducirCancionSiguiente() {  if (indiceCancionActual < canciones.size() - 1) {  indiceCancionActual++;  } else {  indiceCancionActual = 0;  }  reproducirCancion(indiceCancionActual);  }    private void reproducirCancionAleatoria() {  Random random = new Random();  int indiceAleatorio = random.nextInt(canciones.size());  reproducirCancion(indiceAleatorio);  }    private void reproducirCancionEnBucle() {  reproducirCancion(indiceCancionActual);  }    private void reproducirSonidoClic() {  if (clipClic != null) {  clipClic.setFramePosition(0);  clipClic.start();  }  }    private void reproducirCancion(int indice) {  try {  clip.stop();  clip.close();    AudioInputStream inputStream = AudioSystem.getAudioInputStream(MainChess.class.getResourceAsStream("/musica/" + canciones.get(indice)));  clip = AudioSystem.getClip();  clip.open(inputStream);  gainControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER\_GAIN);  gainControl.setValue(volume);  clip.loop(Clip.LOOP\_CONTINUOUSLY);  clip.start();    } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {  e.printStackTrace();  }  }    private static class TransparentLabel extends JLabel {    public TransparentLabel(String text) {  super(text);  setOpaque(false); // Hacer el JLabel transparente  }    @Override  protected void paintComponent(Graphics g) {  // Hacer que el fondo del JLabel sea transparente  Graphics2D g2d = (Graphics2D) g.create();  g2d.setColor(new Color(0, 0, 0, 0)); // Color transparente  g2d.fillRect(0, 0, getWidth(), getHeight());  g2d.dispose();    super.paintComponent(g);  }  }  } |     *Move.java*   |  | | --- | | *package Main;*    *import Pieces.Piece;*    *public class Move {*    *int oldCol;*  *int oldRow;*  *int newCol;*  *int newRow;*    *Piece piece;*  *Piece capture;*    *public Move(Board board, Piece piece, int newCol, int newRow) {*  *this.oldCol = piece.col;*  *this.oldRow = piece.row;*  *this.newCol = newCol;*  *this.newRow = newRow;*    *this.piece = piece;*  *this.capture = board.getPiece(newCol, newRow);*  *}*    *public Move(int col, int row, int col2, int row2, Piece king) {*  *}*  *}* | |

***/Pieces***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Piece.java*   |  | | --- | | package Pieces;    import java.awt.\*;  import javax.imageio.ImageIO;    import Main.Board;    import java.awt.image.BufferedImage;  import java.io.FileInputStream;  import java.io.IOException;    public class Piece {    public int col, row;  public int xPos, yPos;    public boolean isWhite;  public String name;  public int value;    public boolean isFistMove = true;    BufferedImage sheet;  {  try {  sheet = ImageIO.read(new FileInputStream("Chess/src/Images/pieces2.png")); // or "Udistrital\_Java\_Ajedrez/Chess/src/Images/pieces.png"  } catch (IOException e) {  e.printStackTrace();  }  }  protected int getSheetScale = sheet.getWidth()/6;  protected int sheetScale = sheet.getWidth()/6;    Image sprite;  Board board;    public Piece(Board board) {  this.board = board;  }    public boolean isValidMovement(int col, int row) {return true;}  public boolean moveCollidesWithPiece(int col, int row) {return false;}    public void paint(Graphics2D g2d) {  g2d.drawImage(sprite, xPos, yPos, null);  }  } |     *Bishop.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;    public class Bishop extends Piece{  public Bishop(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "Bishop";    this.sprite = sheet.getSubimage(2 \* sheetScale, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    if (col < 0 || col >= 8 || row < 0 || row >= 8) {  return false;  } else if (Math.abs(this.col - col) == Math.abs(this.row - row)) {  return true;  } else {  return false;  }  }    public boolean moveCollidesWithPiece(int col, int row) {    //up left  if (this.col > col && this.row > row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col - i, this.row - i) != null)  return true;    //up right  if (this.col < col && this.row > row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col + i, this.row - i) != null)  return true;    //down left  if (this.col > col && this.row < row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col - i, this.row + i) != null)  return true;    //down right  if (this.col < col && this.row < row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col + i, this.row + i) != null)  return true;      return false;  }  } |     *King.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;  import Main.Move;    public class King extends Piece{  public King(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "King";    this.sprite = sheet.getSubimage(0, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    if (col < 0 || col >= 8 || row < 0 || row >= 8) {  return false;  } else if (Math.abs((col - this.col) \* (row - this.row)) == 1 || Math.abs(col - this.col) + Math.abs(row - this.row) == 1 || canCastle(col, row)) {  return true;  } else {  return false;  }  }      private boolean canCastle(int col, int row) {    if (this.row == row) {    if (col == 6) {  Piece rook = board.getPiece(7, row);  if (rook != null && rook.isFistMove && isFistMove) {  return board.getPiece(5, row) == null &&  board.getPiece(6, row) == null &&  !board.checkScanner.isKingChecked(new Move(board, this, 5, row));  }  } else if (col == 2) {  Piece rook = board.getPiece(0, row);  if (rook != null && rook.isFistMove && isFistMove) {  return board.getPiece(3, row) == null &&  board.getPiece(2, row) == null &&  board.getPiece(1, row) == null &&  !board.checkScanner.isKingChecked(new Move(board, this, 3, row));    }  }  }    return false;  }  } |     *Knight.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;    public class Knight extends Piece{  public Knight(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "Knight";    this.sprite = sheet.getSubimage(3 \* sheetScale, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    if (col < 0 || col >= 8 || row < 0 || row >= 8) {  return false;  } else if (Math.abs(col - this.col) \* Math.abs(row - this.row) == 2) {  return true;  } else {  return false;  }  }  } |     *Pawn.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;    public class Pawn extends Piece{  public Pawn(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "Pawn";    this.sprite = sheet.getSubimage(5 \* sheetScale, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    int colorIndex = isWhite ? 1 : -1;    // push pawn 1  if (this.col == col && row == this.row - colorIndex && board.getPiece(col, row) == null)  return true;    // push pawn 2  if (isFistMove && this.col == col && row == this.row - colorIndex \* 2 && board.getPiece(col, row) == null && board.getPiece(col, row + colorIndex) == null)  return true;    // capture left  if (col == this.col - 1 && row == this.row - colorIndex && board.getPiece(col, row) != null)  return true;    // capture right  if (col == this.col + 1 && row == this.row - colorIndex && board.getPiece(col, row) != null)  return true;    // en passant left  if (board.getTileNum(col, row) == board.enPassantTile && col == this.col - 1 && row == this.row - colorIndex && board.getPiece(col, row + colorIndex) != null) {  return true;  }    // en passant right  if (board.getTileNum(col, row) == board.enPassantTile && col == this.col + 1 && row == this.row - colorIndex && board.getPiece(col, row + colorIndex) != null) {  return true;  }    return false;  }  } |     *Queen.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;    public class Queen extends Piece{  public Queen(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "Queen";    this.sprite = sheet.getSubimage(sheetScale, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    if (col < 0 || col >= 8 || row < 0 || row >= 8) {  return false;  } else if (this.col == col || this.row == row || Math.abs(this.col - col) == Math.abs(this.row - row)) {  return true;  } else {  return false;  }  }    public boolean moveCollidesWithPiece(int col, int row) {    if (this.col == col || this.row == row) {    //left  if (this.col > col)  for (int c = this.col - 1; c > col; c--)  if (board.getPiece(c, this.row) != null)  return true;    //right  if (this.col < col)  for (int c = this.col + 1; c < col; c++)  if (board.getPiece(c, this.row) != null)  return true;    //up  if (this.row > row)  for (int r = this.row - 1; r > row; r--)  if (board.getPiece(this.col, r) != null)  return true;    //down  if (this.row < row)  for (int r = this.row + 1; r < row; r++)  if (board.getPiece(this.col, r) != null)  return true;    } else {    //up left  if (this.col > col && this.row > row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col - i, this.row - i) != null)  return true;    //up right  if (this.col < col && this.row > row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col + i, this.row - i) != null)  return true;    //down left  if (this.col > col && this.row < row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col - i, this.row + i) != null)  return true;    //down right  if (this.col < col && this.row < row)  for (int i = 1; i < Math.abs(this.col - col); i++)  if (board.getPiece(this.col + i, this.row + i) != null)  return true;  }    return false;  }  } |     *Rook.java*   |  | | --- | | package Pieces;    import java.awt.image.BufferedImage;    import Main.Board;    public class Rook extends Piece{  public Rook(Board board, int col, int row, boolean isWhite) {  super(board);  this.col = col;  this.row = row;  this.xPos = col \* board.tileSize;  this.yPos = row \* board.tileSize;    this.isWhite = isWhite;  this.name = "Rook";    this.sprite = sheet.getSubimage(4 \* sheetScale, isWhite ? 0 : sheetScale, sheetScale, sheetScale).getScaledInstance(board.tileSize, board.tileSize, BufferedImage.SCALE\_SMOOTH);  }    public boolean isValidMovement(int col, int row) {    if (col < 0 || col >= 8 || row < 0 || row >= 8) {  return false;  } else if (this.col == col || this.row == row) {  return true;  } else {  return false;  }  }    public boolean moveCollidesWithPiece(int col, int row) {    //left  if (this.col > col)  for (int c = this.col - 1; c > col; c--)  if (board.getPiece(c, this.row) != null)  return true;    //right  if (this.col < col)  for (int c = this.col + 1; c < col; c++)  if (board.getPiece(c, this.row) != null)  return true;    //up  if (this.row > row)  for (int r = this.row - 1; r > row; r--)  if (board.getPiece(this.col, r) != null)  return true;    //down  if (this.row < row)  for (int r = this.row + 1; r < row; r++)  if (board.getPiece(this.col, r) != null)  return true;      return false;  }  } | |

*Clases importadas*

* **java.awt.\***: Este paquete proporciona clases para la creación de interfaces de usuario (UI) y la manipulación de gráficos en Java. Incluye clases para ventanas, componentes, gráficos, colores, eventos, etc.
* **javax.swing.\***: Este paquete es una extensión del paquete **java.awt** y proporciona componentes de interfaz de usuario (UI) adicionales, como botones, etiquetas, paneles, cuadros de diálogo, etc., que son más flexibles y personalizables que los componentes de AWT.
* **java.awt.event.ActionEvent** y **java.awt.event.ActionListener**: Estas clases son parte del paquete **java.awt.event** y se utilizan para manejar eventos de acción en componentes de la interfaz de usuario. **ActionEvent** representa un evento de acción, mientras que **ActionListener** es una interfaz que debe implementarse para recibir y manejar los eventos de acción.
* **java.awt.image.BufferedImage**: Esta clase pertenece al paquete **java.awt.image** y se utiliza para manipular imágenes en formato de mapa de bits. Proporciona métodos para crear, cargar y manipular imágenes, así como para acceder y modificar los píxeles individuales de la imagen.
* **javax.imageio.ImageIO**: Esta clase se encuentra en el paquete **javax.imageio** y se utiliza para leer y escribir imágenes en varios formatos de archivo. Proporciona métodos estáticos para cargar imágenes desde archivos y guardar imágenes en archivos.
* **javax.sound.sampled.\***: Este paquete proporciona clases y interfaces para trabajar con sonidos en Java. Incluye clases para reproducir, grabar y manipular audio en formatos como WAV y MIDI.
* **java.io.File** y **java.io.IOException**: Estas clases son parte del paquete **java.io** y se utilizan para realizar operaciones de entrada/salida en archivos y directorios. **File** representa un archivo o directorio en el sistema de archivos, mientras que **IOException** es una excepción lanzada en caso de error de entrada/salida.
* **java.util.ArrayList** y **java.util.List**: Estas clases son parte del paquete **java.util** y se utilizan para trabajar con listas dinámicas en Java. **ArrayList** es una implementación de la interfaz **List** y proporciona métodos para agregar, eliminar y acceder a elementos en una lista.
* **java.awt.event.MouseAdapter** y **java.awt.event.MouseEvent**: Estas clases son parte del paquete **java.awt.event** y se utilizan para manejar eventos del mouse en la interfaz de usuario. **MouseAdapter** es una clase adaptadora que proporciona implementaciones predeterminadas de los métodos de la interfaz de escucha de eventos del mouse, mientras que **MouseEvent** representa un evento del mouse.
* **java.io.BufferedWriter** y **java.io.FileWriter**: Estas clases son parte del paquete **java.io** y se utilizan para escribir texto en archivos de manera eficiente. **BufferedWriter** es una clase que proporciona un búfer de escritura para mejorar el rendimiento de las operaciones de escritura, y **FileWriter** es una clase que se utiliza para escribir caracteres en un archivo.
* **java.util.Arrays**: Esta clase es parte del paquete **java.util** y proporciona métodos estáticos para manipular matrices (arrays) en Java. Incluye métodos para ordenar, buscar y realizar operaciones de comparación en matrices.
* **java.util.Random**: Esta clase es parte del paquete **java.util** y se utiliza para generar números aleatorios en Java. Proporciona métodos para generar valores aleatorios de diferentes tipos numéricos y establecer una semilla para generar secuencias repetibles de números aleatorios.
* **java.awt.Desktop** y **java.net.URI**: Estas clases son parte de los paquetes **java.awt** y **java.net**, respectivamente, y se utilizan para interactuar con el escritorio y abrir enlaces URI en el navegador web predeterminado. **Desktop** proporciona métodos para abrir archivos, URLs y realizar otras acciones en el entorno de escritorio, mientras que **URI** representa una referencia uniforme de recursos y se utiliza para trabajar con identificadores de recursos uniformes, como enlaces web.

1. Testeo

|  |  |
| --- | --- |
| **Entrada** | En el label poner los nombres para las selección de piezas, además de oprimir en el play para iniciar el juego |
| **Salida** | Inserting image... |

*Casos de prueba (visibles)*

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| Click en la introducción para saltarla |  |

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| La parición del movimiento posible de la pieza cuando esta se mueva |  |

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| La muestra de los nombres de la interface a el juego |  |

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| Generación del .txt |  |

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| Las piezas no se salgan del tablero | Le devolución de las piezas al tablero si se intentan salir o si hacen un movimiento invalido |

|  |  |
| --- | --- |
| **Entradas de ejemplo** | **Salida de ejemplo** |
| Al oprimir los botones se tiene que generar con el de menos decrecer la música, con el de más crecer la música. el del centro pausar, el de la nota musical cambiar la música. el de “x” cerrar la ventana, el de interrogación enviar al video explicativo |  |