

Jogo da Velha em Java

Com conexão em banco de dados

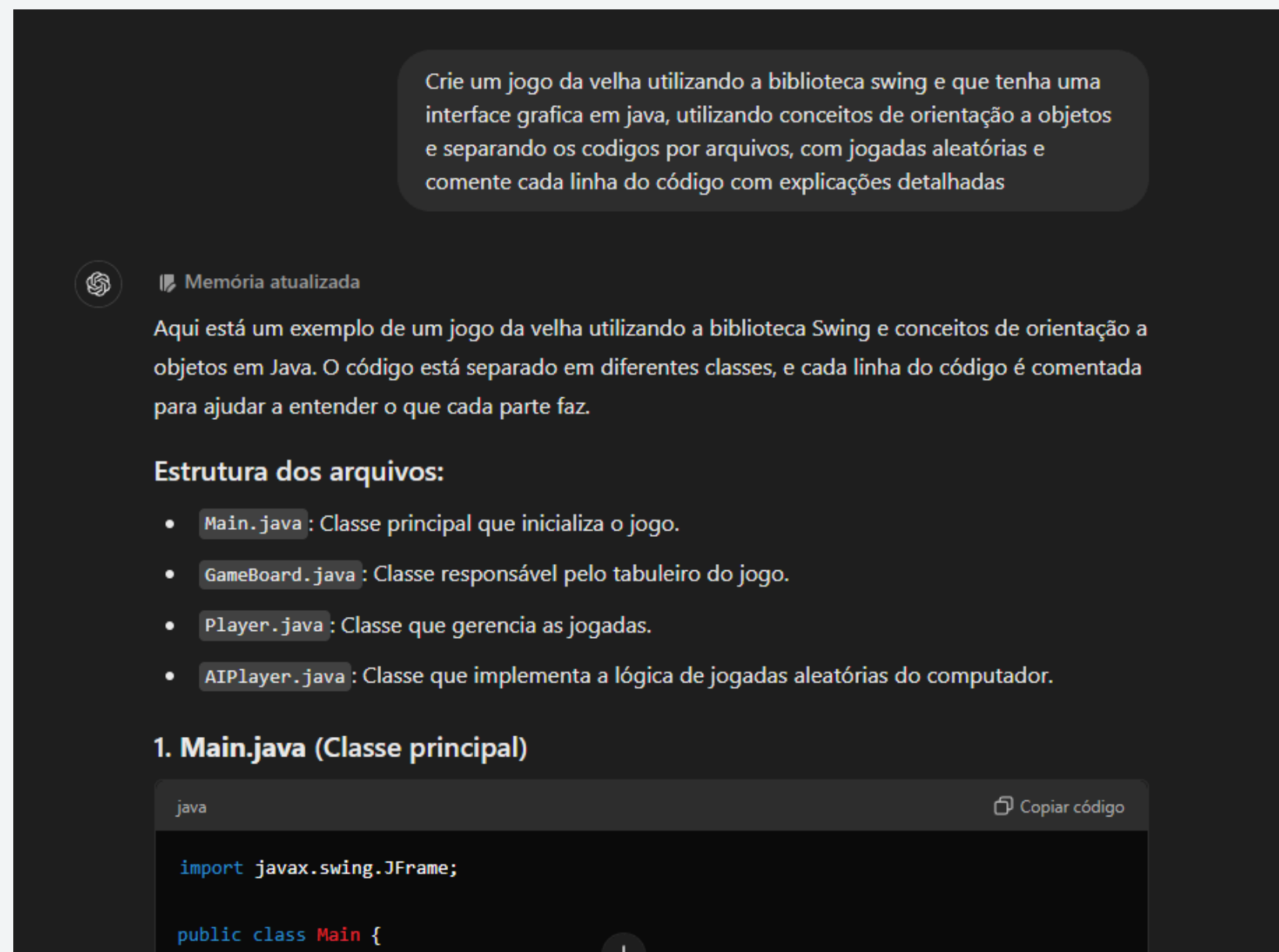


PROCESSO CRIATIVO

Processo Criativo:

Início da modelagem primeiro prompt:

Pedimos ao chat uma prompt muito pouca clara e aberta deixando aberta a possibilidade de erros e coisas do genero:



Processo Criativo:

Primeiro prompt:

4. AIPlayer.java (Jogador Automático)

```
java Copiar código

import java.util.Random;
import javax.swing.JButton;

public class AIPlayer extends Player {
    private Random random;

    public AIPlayer(String symbol) {
        super(symbol); // Chama o construtor da classe Player
        random = new Random(); // Inicializa o gerador de números aleatórios
    }

    // Método que retorna uma jogada aleatória
    public int[] getRandomMove(JButton[][] buttons) {
        int row, col;

        // Gera posições aleatórias até encontrar um botão vazio
        do {
            row = random.nextInt(3); // Gera uma linha aleatória (0-2)
            col = random.nextInt(3); // Gera uma coluna aleatória (0-2)
        } while (!buttons[row][col].getText().equals("")); // Continua até achar um espaço

        return new int[]{row, col}; // Retorna a posição escolhida
    }
}
```

Ao enviarmos esse prompt inicial ao chatgpt recebemos um código de um jogo da velha que, apresentava diversos erros, e buscava utilizar de conceitos muito avançados para realizar o jogo da velha.

Processo Criativo

Create a tic-tac-toe game using the Swing library and with a graphical interface in Java, utilizing object-oriented concepts and separating the code into different files, with random moves and comment each line of the code with detailed explanations.

Tic Tac Toe Game in Java using Swing Library

Below is a simple implementation of a Tic Tac Toe game using Java's Swing library. The game alternates between two players, 'X' and 'O', and includes a feature for the computer to make random moves.

Project Structure

The project is divided into three classes:

- `TicTacToeGame.java`: This class contains the main method and is responsible for creating the game window.
- `GameBoard.java`: This class represents the game board and handles the game logic.
- `GameButton.java`: This class represents a single button on the game board.

TicTacToeGame.java

Em seguida pedimos uma prompt similar ao blackbox porém já com algumas alterações para facilitar a tradução e adaptação do código.

Realizamos a tradução do código e das explicações do blackbox para utilizarmos os dados.

Códigos do blackbox:

1)

```
java Verify Open In Editor

1 // Import necessary libraries
2 import javax.swing.*;
3 import java.awt.*;
4
5 // Main class for the Tic Tac Toe game
6 public class TicTacToeGame {
7     // Create a new JFrame for the game window
8     private JFrame frame;
9
10    // Create a new GameBoard instance
11    private GameBoard gameBoard;
12
13    // Constructor for the TicTacToeGame class
14    public TicTacToeGame() {
15        // Create a new JFrame with a title
16        frame = new JFrame("Tic Tac Toe Game");
17        // Set the default close operation to exit the application when
18        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19        // Create a new GameBoard instance and add it to the frame
20        gameBoard = new GameBoard();
21        frame.add(gameBoard);
22        // Set the frame's size to 300x300 pixels
23        frame.setSize(300, 300);
24        // Center the frame on the screen
25        frame.setLocationRelativeTo(null);
26        // Make the frame visible
27        frame.setVisible(true);
28    }
29
```

2)

```
23     frame.setSize(300, 300);
24     // Center the frame on the screen
25     frame.setLocationRelativeTo(null);
26     // Make the frame visible
27     frame.setVisible(true);
28 }
29
30 // Main method to start the game
31 public static void main(String[] args) {
32     // Create a new instance of the TicTacToeGame class
33     new TicTacToeGame();
34 }
35 }
```

Parte principal do jogo da velha

Códigos do blackbox:

Tabuleiro do jogo da velha:

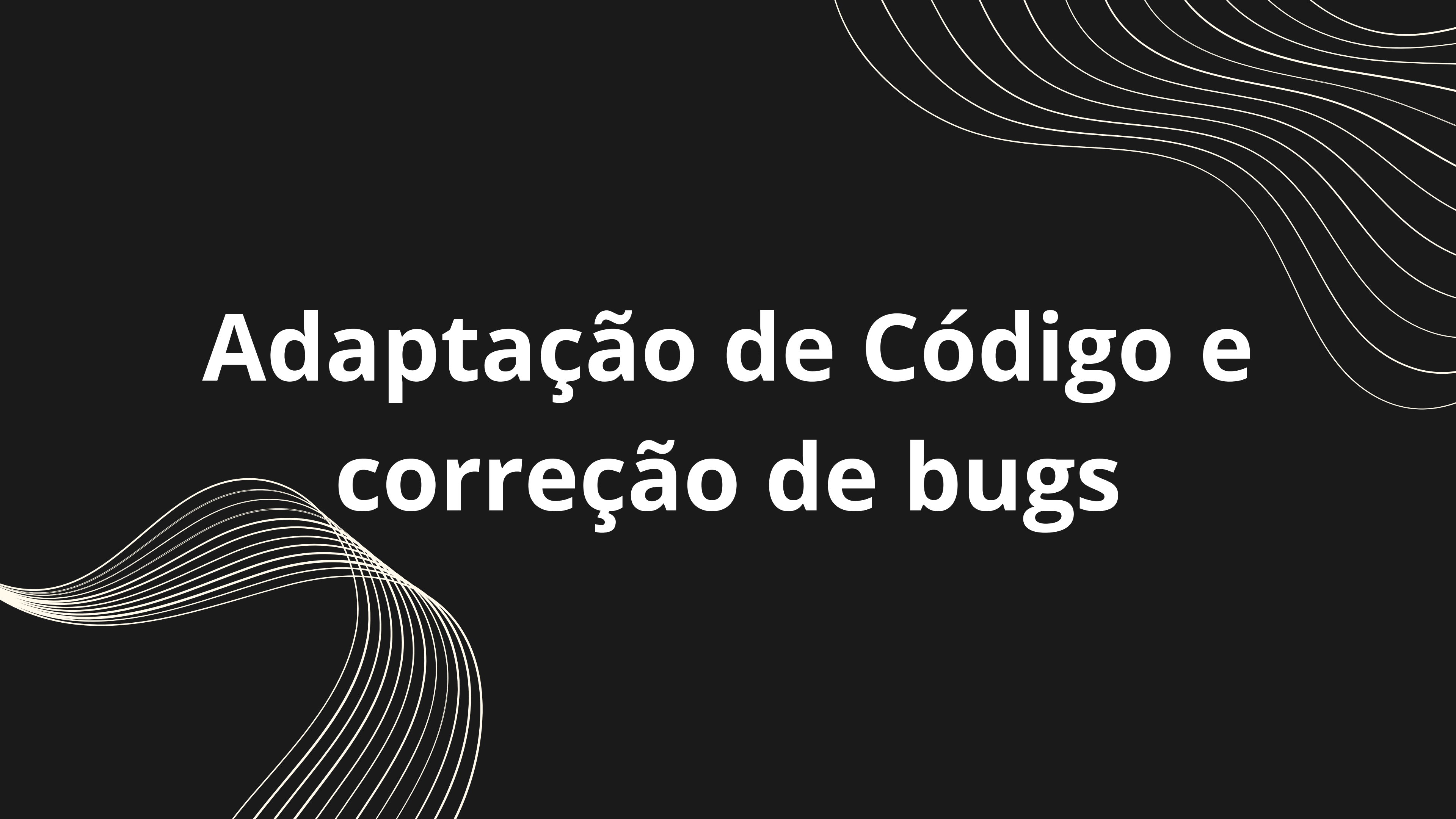
```
1 // Import necessary libraries
2 import javax.swing.*;
3 import java.awt.*;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.util.Random;
7
8 // Class representing the game board
9 public class GameBoard extends JPanel {
10     // Create a 3x3 array of GameButton instances
11     private GameButton[][] buttons = new GameButton[3][3];
12     // Variable to keep track of whose turn it is
13     private boolean isPlayerTurn = true;
14     // Variable to keep track of the number of moves made
15     private int movesMade = 0;
16
17     // Constructor for the GameBoard class
18     public GameBoard() {
19         // Set the layout of the panel to a 3x3 grid
20         setLayout(new GridLayout(3, 3));
21         // Create a new GameButton instance for each position on the board
22         for (int i = 0; i < 3; i++) {
23             for (int j = 0; j < 3; j++) {
24                 buttons[i][j] = new GameButton(i, j);
25                 // Add an action listener to each button
26                 buttons[i][j].addActionListener(new ButtonClickListener());
27                 // Add the button to the panel
28                 add(buttons[i][j]);
29             }
30         }
31     }
32
33     // Inner class representing an action listener for the buttons
34     private class ButtonClickListener implements ActionListener {
35         // Method to handle button clicks
36         public void actionPerformed(ActionEvent e) {
37             // Get the button that was clicked
38             GameButton button = (GameButton) e.getSource();
39             // If the button is empty and it's the player's turn
40             if (button.isEmpty() && isPlayerTurn) {
41                 // Set the button's text to 'X'
42                 button.setText("X");
43                 // Increment the number of moves made
44                 movesMade++;
45                 // Check if the player has won
46                 if (checkForWin('X')) {
47                     // If the player has won, display a message and exit the game
48                     JOptionPane.showMessageDialog(null, "Player wins!");
49                     System.exit(0);
50                 }
51                 // If the player hasn't won and all squares are filled,
52                 else if (movesMade == 9) {
53                     JOptionPane.showMessageDialog(null, "It's a draw!");
54                     System.exit(0);
55                 }
56                 // If the player hasn't won and there are still empty squares,
57                 else {
58                     // Make the computer's move
59                     makeComputerMove();
60                 }
61             }
62         }
63     }
64
65     // Method to make the computer's move
66     private void makeComputerMove() {
67         // Create a new Random instance
68         Random random = new Random();
69         // Variable to store the row and column of the computer's move
70         int row, col;
71         // Loop until an empty square is found
72         do {
73             // Generate random row and column indices
74             row = random.nextInt(3);
75             col = random.nextInt(3);
76         } while (!buttons[row][col].isEmpty());
77         // Set the text of the button at the generated row and column to 'O'
78         buttons[row][col].setText("O");
79         // Increment the number of moves made
80         movesMade++;
81         // Check if the computer has won
82         if (checkForWin('O')) {
83             // If the computer has won, display a message and exit the game
84             JOptionPane.showMessageDialog(null, "Computer wins!");
85             System.exit(0);
86         }
87     }
88 }
```

Códigos do blackbox:

```
80     movesMade++;
81     // Check if the computer has won
82     if (checkForWin('0')) {
83         // If the computer has won, display a message and exit the game
84         JOptionPane.showMessageDialog(null, "Computer wins!");
85         System.exit(0);
86     }
87     // If the computer hasn't won and all squares are filled, it's a draw
88     else if (movesMade == 9) {
89         JOptionPane.showMessageDialog(null, "It's a draw!");
90         System.exit(0);
91     }
92     // If the computer hasn't won and there are still empty squares,
93     else {
94         // Switch turns
95         isPlayerTurn = true;
96     }
97 }
98
99 // Method to check if a player has won
100 private boolean checkForWin(char player) {
101     // Check rows and columns for a win
102     for (int i = 0; i < 3; i++) {
103         if (buttons[i][0].getText().equals(String.valueOf(player)) &&
104             buttons[i][1].getText().equals(String.valueOf(player)) &&
105             buttons[i][2].getText().equals(String.valueOf(player))) {
106             return true;
107         }
108         if (buttons[0][i].getText().equals(String.valueOf(player)) &&
109             buttons[1][i].getText().equals(String.valueOf(player)) &&
110             buttons[2][i].getText().equals(String.valueOf(player))) {
111             return true;
112         }
113     }
```

```
103         if (buttons[i][0].getText().equals(String.valueOf(player)) &&
104             buttons[i][1].getText().equals(String.valueOf(player)) &&
105             buttons[i][2].getText().equals(String.valueOf(player))) {
106             return true;
107         }
108         if (buttons[0][i].getText().equals(String.valueOf(player)) &&
109             buttons[1][i].getText().equals(String.valueOf(player)) &&
110             buttons[2][i].getText().equals(String.valueOf(player))) {
111             return true;
112         }
113     }
114     // Check diagonals for a win
115     if ((buttons[0][0].getText().equals(String.valueOf(player)) &&
116         buttons[1][1].getText().equals(String.valueOf(player)) &&
117         buttons[2][2].getText().equals(String.valueOf(player))) ||
118         (buttons[0][2].getText().equals(String.valueOf(player)) &&
119         buttons[1][1].getText().equals(String.valueOf(player)) &&
120         buttons[2][0].getText().equals(String.valueOf(player)))) {
121         return true;
122     }
123     // If no win is found, return false
124     return false;
125 }
126 }
```

Funcionamento e detecção de vencedor

The background features a solid black field with two sets of thin, white, wavy lines. One set of lines flows from the top right corner towards the center, while another set flows from the bottom left corner towards the center, creating a sense of movement and depth.

Adaptação de Código e correção de bugs

Adaptação de Código:

Pegue esse código como base e finja que você é um estudante e acabou de aprender conceitos básicos de programação orientada a objetos, pelo menos faça o jogo da velha funcionar do jeito antigo, mantenha as linhas de comentário q ainda servirem e comente as novas

```
// Importar bibliotecas necessárias
import javax.swing.*;
import java.awt.*;

// Classe principal para o jogo da Velha
public class JogoDaVelha {
    // Criar um novo JFrame para a janela do jogo
    private JFrame janela;

    // Criar uma nova instância de Tabuleiro
    private Tabuleiro tabuleiro;

    // Construtor da classe JogoDaVelha
    public JogoDaVelha() {
        // Criar um novo JFrame com um título
        janela = new JFrame("Jogo da Velha");
        // Definir a operação padrão de fechamento para sair do
        aplicativo ao fechar a janela
        janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Criar uma nova instância de Tabuleiro e adicioná-la à janela
        tabuleiro = new Tabuleiro();
        janela.add(tabuleiro);
        // Definir o tamanho da janela para 300x300 pixels
        janela.setSize(300, 300);
    }
}
```

Encontramos o seguinte problema em nossos prompts e códigos: o Blackbox nos passou códigos avançados demais e com elementos desnecessários ao nosso objetivo. Portanto, tivemos que adaptar nosso código para o que gostaríamos.

Correção de bugs:

erro do sql apresentado no console em execução do netbeans:

No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha
No suitable driver found for jdbc:mysql://localhost:3306/jogovelha

pegue todo o código que me forneceu, analise e fale aonde nele coloco as novas configurações do último arquivo que me mandou e o que exatamente eu preciso fazer para fazer a conexão com o banco de dados



Para integrar o seu jogo da velha em Java com um banco de dados MySQL, você precisa fazer algumas configurações e ajustes no seu código. Vou te dar um guia passo a passo sobre onde e como fazer isso:



O erro "No suitable driver found for jdbc:mysql://localhost:3306/jogovelha" indica que o seu projeto Java não está conseguindo encontrar o driver JDBC necessário para se conectar ao banco de dados MySQL. Isso geralmente acontece por um dos seguintes motivos:

1. **Driver JDBC não incluído:** O driver JDBC do MySQL não está adicionado ao seu projeto.
2. **Classe do driver não carregada:** A classe do driver não está sendo carregada corretamente.

Como Resolver o Problema

1. Baixar o Driver JDBC do MySQL

Se você ainda não tem o driver JDBC do MySQL, você pode baixá-lo do site oficial. O arquivo que você precisa é geralmente chamado `mysql-connector-java-<version>.jar`.

Você pode encontrar o driver no seguinte link:

- [MySQL Connector/J Downloads](#)



Durante o processo de adaptação do código, nos deparamos com alguns problemas. Aqui estão os prompts que usamos para entender os erros e corrigi-los.



Códigos Finais:



Código Final:

Main.java:

```
package jogodavelha;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Cria uma nova instância da GUI do Jogo da Velha
        new JogoDaVelhaGUI(); // Inicia a interface gráfica do jogo
    }
}
```

Banco de dados

```
package jogodavelha;

import java.sql.*;

public class BancoDeDados {
    private Connection connection; // Conexão com o banco de dados

    // Construtor: inicializa a conexão com o banco de dados
    public BancoDeDados() {
        try {
            connection = DriverManager.getConnection(url:"jdbc:mysql://localhost/jogovelha", user:"root", password:"fuckfuckMRpresident27@"); // Substitua 'usuario' e 'senha' pelos seus dados
        } catch (SQLException e) {
            e.printStackTrace(); // Imprime a pilha de erros se ocorrer uma exceção
        }
    }

    // Método para salvar uma partida no banco de dados
    public void salvarPartida(String vencedor, String tabuleiro) {
        String sql = "INSERT INTO partidas (vencedor, tabuleiro) VALUES (?, ?)"; // Comando SQL para inserir dados
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString(parameterIndex:1, vencedor); // Define o vencedor
            statement.setString(parameterIndex:2, tabuleiro); // Define o estado do tabuleiro
            statement.executeUpdate(); // Executa a inserção
        } catch (SQLException e) {
            e.printStackTrace(); // Imprime a pilha de erros se ocorrer uma exceção
        }
    }
}
```

Banco de dados

```
// Método para obter a pontuação do banco de dados
public int[] obterPontuacao() {
    int[] pontuacoes = new int[2]; // Array para armazenar as pontuações do jogador e do computador
    String sql = "SELECT COUNT(*) FROM partidas WHERE vencedor = 'Jogador'"; // SQL para contar as vitórias do jogador
    try (Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(sql)) {
        if (resultSet.next()) {
            pontuacoes[0] = resultSet.getInt(columnIndex:1); // Armazena a pontuação do jogador
        }
    } catch (SQLException e) {
        e.printStackTrace(); // Imprime a pilha de erros se ocorrer uma exceção
    }

    sql = "SELECT COUNT(*) FROM partidas WHERE vencedor = 'Computador'"; // SQL para contar as vitórias do computador
    try (Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(sql)) {
        if (resultSet.next()) {
            pontuacoes[1] = resultSet.getInt(columnIndex:1); // Armazena a pontuação do computador
        }
    } catch (SQLException e) {
        e.printStackTrace(); // Imprime a pilha de erros se ocorrer uma exceção
    }

    return pontuacoes; // Retorna o array com as pontuações
}
}
```

Jogo da Velha

```
package jogodavelha;

import java.util.Random;

public class JogoDaVelha {
    private char[][] tabuleiro; // Matriz que representa o tabuleiro
    private char jogadorAtual; // Símbolo do jogador atual (X ou O)
    private char computador; // Símbolo do computador (O)

    // Construtor: inicializa o jogo e define o jogador e o computador
    public JogoDaVelha() {
        tabuleiro = new char[3][3];
        reiniciaJogo();
        jogadorAtual = 'X';
        computador = 'O';
    }

    // Método para obter o símbolo do computador
    public char getComputador() {
        return computador;
    }

    // Método para realizar a jogada do jogador
    public boolean realizaJogada(int linha, int coluna) {
        if (tabuleiro[linha][coluna] == ' ') { // Verifica se a célula está vazia
            tabuleiro[linha][coluna] = jogadorAtual; // Coloca o símbolo do jogador
            return true;
        }
        return false;
    }

    // Método para realizar a jogada do computador
    public boolean jogadaDoComputador() {
        Random random = new Random();
        int linha, coluna;
        while (true) {
            linha = random.nextInt(bound:3); // Gera uma linha aleatória
            coluna = random.nextInt(bound:3); // Gera uma coluna aleatória
            if (tabuleiro[linha][coluna] == ' ') { // Verifica se a célula está vazia
                tabuleiro[linha][coluna] = computador; // Coloca o símbolo do computador
                return true;
            }
        }
    }
}
```


Jogo da Velha

```
// Método para verificar se um jogador venceu
public boolean verificaVencedor(char simbolo) {
    for (int i = 0; i < 3; i++) {
        // Verifica linhas e colunas
        if ((tabuleiro[i][0] == simbolo && tabuleiro[i][1] == simbolo && tabuleiro[i][2] == simbolo) ||
            (tabuleiro[0][i] == simbolo && tabuleiro[1][i] == simbolo && tabuleiro[2][i] == simbolo)) {
            return true;
        }
    }
    // Verifica as diagonais
    return (tabuleiro[0][0] == simbolo && tabuleiro[1][1] == simbolo && tabuleiro[2][2] == simbolo) ||
           (tabuleiro[0][2] == simbolo && tabuleiro[1][1] == simbolo && tabuleiro[2][0] == simbolo);
}

// Método para verificar se o tabuleiro está cheio
public boolean tabuleiroCheio() {
    for (char[] linha : tabuleiro) {
        for (char celula : linha) {
            if (celula == ' ') { // Retorna falso se encontrar uma célula vazia
                return false;
            }
        }
    }
    return true;
}

// Método para reiniciar o jogo, limpando o tabuleiro
public void reiniciaJogo() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            tabuleiro[i][j] = ' '; // Define todas as células como vazias
        }
    }
    jogadorAtual = 'X'; // Reinicia o jogador atual para 'X'
}
```

```
public char getJogadorAtual() {
    return jogadorAtual;
}

// Alterna o jogador atual entre 'X' e 'O'
public void trocaJogador() {
    jogadorAtual = (jogadorAtual == 'X') ? 'O' : 'X';
}

public char[][] getTabuleiro() {
    return tabuleiro;
}
}
```

Jogo da Velha GUI

```
package jogodavelha;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class JogoDaVelhaGUI {
    private JFrame frame; // Janela principal do jogo
    private JButton[][] botoes = new JButton[3][3]; // Botões que representam o tabuleiro
    private JogoDaVelha jogo = new JogoDaVelha(); // Instância do jogo
    private boolean jogadorAtivo = true; // Indica se o jogador está ativo
    private BancoDeDados bancoDeDados = new BancoDeDados(); // Instância do banco de dados
    private JLabel scoreLabel; // Label para exibir a pontuação

    // Construtor: cria a interface gráfica do jogo
    public JogoDaVelhaGUI() {
        criaGUI(); // Chama o método para criar a GUI
        atualizarPontuacao(); // Atualiza a pontuação inicial
    }
}
```

Jogo da Velha GUI

```
// Método para criar a interface gráfica
public void criaGUI() {
    frame = new JFrame(title:"Jogo da Velha"); // Cria a janela do jogo
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Fecha o programa ao fechar a janela
    frame.setSize(width:400, height:450); // Define o tamanho da janela
    frame.setLayout(new BorderLayout()); // Define o layout da janela

    JPanel tabuleiroPanel = new JPanel(new GridLayout(rows:3, cols:3)); // Pannel para o tabuleiro

    // Criação dos botões do tabuleiro
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            botoes[i][j] = new JButton(text:" "); // Botão vazio
            botoes[i][j].setFont(new Font(name:"Arial", Font.PLAIN, size:60)); // Define a fonte do botão
            final int linha = i; // Captura a linha
            final int coluna = j; // Captura a coluna
            botoes[i][j].addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    if (botoes[linha][coluna].getText().equals(anObject:" ") && jogadorAtivo) {
                        if (jogo.realizaJogada(linha, coluna)) { // Realiza a jogada
                            botoes[linha][coluna].setText(String.valueOf(jogo.getJogadorAtual())); // Atualiza o botão com o símbolo do jogador
                            if (jogo.verificaVencedor(jogo.getJogadorAtual())) {
                                JOptionPane.showMessageDialog(frame, message:"Você ganhou!"); // Mensagem de vitória
                                bancoDeDados.salvarPartida(vencedor:"Jogador", obterEstadoTabuleiro()); // Salva a partida no banco de dados
                                atualizarPontuacao(); // Atualiza a pontuação
                                reiniciaJogo(); // Reinicia o jogo
                            } else if (jogo.tabuleiroCheio()) {
                                JOptionPane.showMessageDialog(frame, message:"Empate!"); // Mensagem de empate
                                bancoDeDados.salvarPartida(vencedor:"Empate", obterEstadoTabuleiro()); // Salva a partida como empate
                                atualizarPontuacao(); // Atualiza a pontuação
                                reiniciaJogo(); // Reinicia o jogo
                            } else {
                                jogadorAtivo = false; // Alterna para o computador
                                jogadaDoComputador(); // Jogada do computador
                            }
                        }
                    }
                }
            });
            tabuleiroPanel.add(botoes[i][j]); // Adiciona o botão ao painel
        }
    }
}
```

```
scoreLabel = new JLabel(text:"Jogador: 0 | Computador: 0"); // Inicializa a label de pontuação
scoreLabel.setFont(new Font(name:"Arial", Font.PLAIN, size:18)); // Define a fonte da label
scoreLabel.setHorizontalAlignment(SwingConstants.CENTER); // Centraliza a label

frame.add(scoreLabel, BorderLayout.NORTH); // Adiciona a label ao topo da janela
frame.add(tabuleiroPanel, BorderLayout.CENTER); // Adiciona o painel do tabuleiro ao centro da janela
frame.setVisible(b:true); // Torna a janela visível
}

// Método para realizar a jogada do computador
private void jogadaDoComputador() {
    try {
        Thread.sleep(millis:250); // Pausa para simular o tempo de resposta do computador
    } catch (InterruptedException ex) {
        ex.printStackTrace(); // Imprime a pilha de erros se ocorrer uma exceção
    }

    if (jogo.jogadaDoComputador()) { // Se o computador consegue jogar
        atualizarTabuleiro(); // Atualiza o tabuleiro na interface
        if (jogo.verificaVencedor(jogo.getComputador())) {
            JOptionPane.showMessageDialog(frame, message:"Computador ganhou!"); // Mensagem de derrota do jogador
            bancoDeDados.salvarPartida(vencedor:"Computador", obterEstadoTabuleiro()); // Salva a partida no banco de dados
            atualizarPontuacao(); // Atualiza a pontuação
            reiniciaJogo(); // Reinicia o jogo
        } else if (jogo.tabuleiroCheio()) {
            JOptionPane.showMessageDialog(frame, message:"Empate!"); // Mensagem de empate
            bancoDeDados.salvarPartida(vencedor:"Empate", obterEstadoTabuleiro()); // Salva a partida como empate
            atualizarPontuacao(); // Atualiza a pontuação
            reiniciaJogo(); // Reinicia o jogo
        }
        jogadorAtivo = true; // Alterna para o jogador
    }
}
```

Jogo da Velha GUI

```
// Método para reiniciar o jogo
private void reiniciaJogo() {
    jogo.reiniciaJogo(); // Reinicia o jogo
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            botoes[i][j].setText(text: " "); // Limpa todos os botões
        }
    }
    atualizarPontuacao(); // Atualiza a pontuação ao reiniciar
}

// Método para atualizar o tabuleiro na interface
private void atualizarTabuleiro() {
    char[][] tabuleiro = jogo.getTabuleiro(); // Obtém o estado atual do tabuleiro
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            botoes[i][j].setText(String.valueOf(tabuleiro[i][j])); // Atualiza o botão com o valor do tabuleiro
        }
    }
}

// Método para obter o estado do tabuleiro em formato de String
private String obterEstadoTabuleiro() {
    StringBuilder sb = new StringBuilder();
    char[][] tabuleiro = jogo.getTabuleiro(); // Obtém o estado atual do tabuleiro
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            sb.append(tabuleiro[i][j]); // Adiciona o valor ao StringBuilder
        }
    }
    return sb.toString(); // Retorna a representação do tabuleiro
}

// Método para atualizar a pontuação
private void atualizarPontuacao() {
    int[] pontuacoes = bancoDeDados.obterPontuacao(); // Obtém a pontuação atual do banco de dados
    scoreLabel.setText("Jogador: " + pontuacoes[0] + " | Computador: " + pontuacoes[1]); // Atualiza a label com as pontuações
}

// Método principal para executar o jogo
Run | Debug
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new JogoDaVelhaGUI(); // Cria uma nova instância do jogo
        }
    });
}
```


Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany</groupId>
  <artifactId>JogoDaVelhaGUI</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version> <!-- Verifique se há uma versão mais recente -->
    </dependency>
  </dependencies>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>23</maven.compiler.release>
    <exec.mainClass>com.mycompany.jogodavelhagui.JogoDaVelhaGUI</exec.mainClass>
  </properties>
</project>
```

Script do SQL

```
CREATE DATABASE IF NOT EXISTS jogovelha;  
USE jogovelha;
```

```
) CREATE TABLE IF NOT EXISTS partidas (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    vencedor VARCHAR(50) NOT NULL,  
    tabuleiro TEXT NOT NULL  
);
```

```
SELECT * FROM partidas;
```