
**UNIVERSIDADE FEDERAL DE OURO PRETO**
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO - DECOM
**Professor: GUILHERMO CAMARA CHAVEZ**

# Desenvolvimento do jogo TERMO em Java com Interface Gráfica (Swing)

**Disciplina: BCC 221 - Programação Orientada a Objetos (POO)  Turma: 11**

João Pedro dos Santos Ferraz 23.1.4030
João Vitor Coelho Oliveira 23.1.4133
Luiza Oliveira Magalhães de Souza 23.1.4009

**OURO PRETO**
**2025**

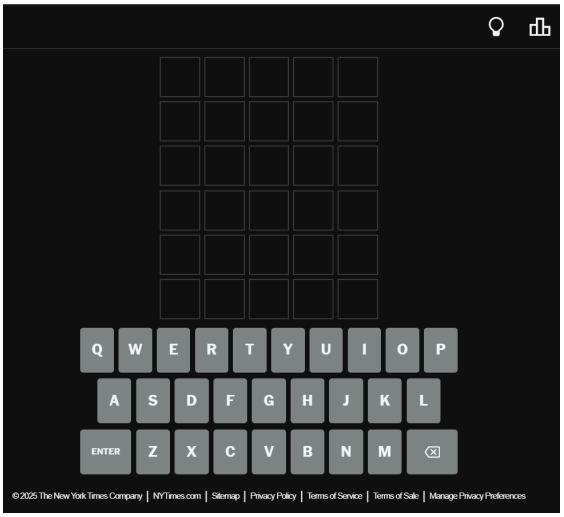
# Sumário

**Sumário** **2**
**Introdução** **3**
Apresentação geral do problema abordado 3
Objetivos do trabalho 3
Objetivos Específicos 3
Escopo do sistema desenvolvido 3
Instruções de compilação e execução 4
**Desenvolvimento** **4**
A tela do jogo 4
A tela do jogo 5
**Descrição do Sistema** **11**
Principais Funcionalidade 11
Entidades Manipuladas 11
Interação com o Usuário 12
**Decisões de Projeto** **12**
Modelagem 12
Estruturas da Biblioteca Swing 12
Persistência de Dados 13
**Arquitetura do Projeto** **13**
Descrição dos Arquivos 13
Relacionamento entre Classes 13
Diagramas de Fluxo 14
Diagramas UML 15
Testes Realizados 16
Tratamento de Erros 19
Implementação do ponto extra 20
**Conclusão** **21**

# Introdução

## Apresentação geral do problema abordado

Este relatório aborda sobre o desenvolvimento de uma versão do jogo TERMO em Java, utilizando a biblioteca Swing para a interface gráfica. O objetivo do jogo é descobrir uma palavra secreta de 5 letras em até 6 tentativas. Para cada palpite, o jogador recebe dicas visuais sobre as letras: verde para a posição correta, amarelo para a posição errada, mas na palavra, e cinza para letras que não fazem parte da palavra.





(Wordle // Termo)

## Objetivos do trabalho

O principal objetivo do trabalho é aplicar de forma prática os conceitos de Programação Orientada a Objetos (POO). Entre os conceitos aplicados estão classes e objetos, encapsulamento, associação entre classes, herança e polimorfismo, e manipulação de arquivos. O projeto também foca no uso prático da biblioteca Swing para a criação de uma interface gráfica.

## Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Implementação do Termo em Java.
- Explorar conceitos da Programação Orientada a Objetos em Java sendo polimorfismo, heranças e também, agora, a adição de interfaces gráficas com Swing.

## Escopo do sistema desenvolvido

O objetivo principal deste trabalho é desenvolver uma versão do jogo Termo, utilizando a

interface gráfica da linguagem Java. O jogo deve possuir um sistema de pontuação que permita salvar os dados de um determinado jogador em um arquivo, assim guardando o progresso da quantidade de jogos iniciados e a quantidade de palavras corretas e erradas de cada jogador.

Além disso, seguindo o funcionamento do Termo, foi implementado um sistema para gerenciar a palavra a ser descoberta, que é escolhida aleatoriamente dentre 1000 opções de um arquivo .txt. As palavras têm, obrigatoriamente, 5 letras no máximo. Pela forma como foi implementado, não é possível usar acentos.

Usando um teclado virtual, o jogador terá que fazer até 6 tentativas para descobrir a palavra secreta, usando dicas visuais para tentar acertar a palavra antes de acabar suas chances. A cada palpite, o jogo informará, através de cores, quais letras pertencem à palavra secreta, quais não pertencem e quais pertencem, mas não estão na posição correta. As letras que não pertencem à palavra misteriosa são desativadas e não podem mais ser escolhidas.

Ao final, se o jogador acertar todas as letras da palavra, ele será vitorioso, e uma mensagem de parabéns será exibida. Por outro lado, caso o número máximo de erros seja atingido, o jogador será informado da derrota. Em ambos os cenários, uma mensagem aparecerá detalhando as estatísticas do jogador.

Como no jogo Termo original, não tem suporte para teclado físico, sendo possível jogar apenas pela interface do SWING.

### Instruções de compilação e execução

```
Java

Compilação: javac -d bin src/com/termo/*.java

Execução: java -cp bin com.termo.Main palavras.txt
```

## Desenvolvimento

### A tela do jogo

- Janela de dados do usuário;
- Tela principal;
- Teclado virtual;
- Grade 5x6 com;
- Painel horizontal com informações da partida e do jogador;
- Painel lateral com botões de:
    - Nova partida;
    - Sair do jogo;

A tela do jogo

Na implementação do trabalho, foram criadas algumas classes, sendo elas :

- **Main**
- **Interface**
- **BancoPalavras**
- **Jogo**
- **Jogador**
- **Estatisticas**

Vamos abordar individualmente cada uma, seu funcionamento e sua correlação.

A classe **Main** é a principal, pois dá início ao nosso jogo. Ela carrega os dados dos jogadores de um arquivo (jogadores.csv) e pede para o usuário fazer login ou criar um novo perfil. Em seguida, ela cria uma nova instância da classe Interface (a janela do jogo) e a torna visível para o jogador.

Java

```java
public class Main {
    // Mapa que armazena todos os jogadores, usando o nome como
chave.
    private static final Map<String, Jogador> jogadores = new
HashMap<>();
    // O nome do arquivo onde os dados dos jogadores são salvos.
    private static final String CAMINHO_JOGADORES =
"jogadores.csv";

    // O método principal que é executado quando o programa é
iniciado.
    public static void main(String[] args) {
        // Verifica se o caminho para o arquivo de palavras foi
fornecido.
        if (args.length < 1) {
            // Se não houver, exibe uma mensagem de erro e fecha
o programa.
            System.err.println("Uso: java -cp bin com.termo.Main
<caminho-para-banco-de-palavras.txt>");
```

```
            System.exit(1);
        }
        // Armazena o caminho do arquivo de palavras.
        final String caminhoPalavras = args[0];
        // Carrega os dados dos jogadores de um arquivo.
        carregarJogadores();
        // Adiciona um "gancho" pra executar automaticamente
quando o programa for fechado.
        // A função é salvar os dados dos jogadores para não
perder o progresso.
        Runtime.getRuntime().addShutdownHook(new
Thread(Main::salvarJogadores));
        // Executa o processo de login ou criação de um novo
jogador.
        Jogador jogadorLogado = executarLoginSimplificado();

        // Se o login for bem-sucedido (o jogador não é nulo).
        if (jogadorLogado != null) {
            // Inicia a interface gráfica (a janela do jogo).
            SwingUtilities.invokeLater(() -> {
                // Cria uma nova interface do jogo e a torna
visível.
                Interface gui = new Interface(caminhoPalavras,
jogadorLogado);
                gui.setVisible(true);
            });
        } else {
            // Se o login for cancelado, fecha o programa.
            System.exit(0);
        }
    }
```

**Interface.Java:** Esta classe é a interface gráfica (GUI) do jogo. Ela é responsável por desenhar a grade de letras, o teclado virtual e os rótulos que mostram as estatísticas do

jogador. A interface se comunica com as outras para pegar uma nova palavra secreta do ***BancoPalavras***; enviar o palpite do jogador para a classe ***Jogo*** para ser avaliado; atualizar os rótulos de estatísticas com os dados da classe ***Jogador.***

Java

```
public class Interface extends JFrame {
    private static final Color COR_CORRETO = new Color(6990436);
    private static final Color COR_PRESENTE = new Color(13218904);
    private static final Color COR_AUSENTE = new Color(7896190);
    private static final Color COR_PADRAO_TECLA = new
Color(13883098);
    private static final Color COR_BORDA_CELULA = new
Color(8882828);
    private final Bancopalavras bancoPalavras;
    private Jogo jogo;
    private final Jogador jogadorLogado;
    private final JLabel[][] celulas = new JLabel[6][5];
    private int linhaAtual = 0;
    private int colunaAtual = 0;
    private final JLabel lblUsuario = new JLabel();
    private final JLabel lblJogos = new JLabel("Jogos: 0");
    private final JLabel lblVitorias = new JLabel("Vitórias: 0");
    private final JLabel lblDerrotas = new JLabel("Derrotas: 0");
    private final JLabel lblSeqVitorias = new JLabel("Seq.
Vitórias: 0");
    private final JLabel lblMelhorSeq = new JLabel("Melhor Seq.:
0");
    private final JLabel lblTentativas = new JLabel("Tentativas
restantes: 6");
    private final Map<Character, JButton> mapaTeclado = new
HashMap();
    private final Map<Character, EstadoTecla> estadoTeclas = new
HashMap();
    private final JButton btnEnter = new JButton("ENTER");
    private final JButton btnApagar = new JButton("APAGAR");
```

**Bancopalavras.java:** Esta classe é o "dicionário" do jogo. Ela lê e armazena uma lista de palavras de um arquivo de texto. Sua principal função é fornecer uma nova palavra secreta aleatória para cada partida e verificar se um palpite do jogador existe no banco de palavras.

Java

```
public class Bancopalavras {
   private final List<String> palavras = new ArrayList();
   private final Random aleatorio = new Random();
   private final Set<String> palavrasUsadas = new HashSet();

   public Bancopalavras(String var1) throws IOException {
      List var2 = Files.readAllLines(Paths.get(var1),
StandardCharsets.UTF_8);
      Iterator var3 = var2.iterator();

      while(var3.hasNext()) {
         String var4 = (String)var3.next();
         String var5 = var4.trim().toLowerCase();
         if (var5.length() == 5) {
            this.palavras.add(var5);
         }
      }

      if (this.palavras.isEmpty()) {
         throw new IOException("Banco de palavras vazio ou
inválido: " + var1);
      }
   }
```

**`Jogo.java`:** Esta classe contém a lógica central do jogo. Ela guarda a palavra secreta e o número de tentativas restantes. O método mais importante é o avaliar(), que compara o palpite do jogador com a palavra secreta e determina se cada letra está na posição correta, na palavra, ou ausente.

Java

```java
public class Jogo {
    private final String palavraSecreta;
    private int tentativasRestantes = 6;
    private final List<String> palpites = new ArrayList();

    public Jogo(String var1) {
        this.palavraSecreta = var1.toLowerCase();
    }
    public String getPalavraSecreta() {
        return this.palavraSecreta;
    }
    public int getTentativasRestantes() {
        return this.tentativasRestantes;
    }
}
```

**`Jogador.java`:** Esta classe representa o perfil de um jogador. Ela armazena o nome do jogador, suas estatísticas de jogo e sua sequência de vitórias. Ela também tem a função de converter seus dados para uma linha de texto (CSV) para que possam ser salvos em um arquivo.

Java

```java
public class Jogador {
    private final String nome;
    private final Estatisticas estatisticas;
    private int sequenciaVitoriasAtual;
    private int melhorSequencia;
    public Jogador(String var1) {
        this.nome = var1;
        this.estatisticas = new Estatisticas();
        this.sequenciaVitoriasAtual = 0;
        this.melhorSequencia = 0;
```

```
    }
    public Jogador(String var1, int var2, int var3, int var4, int
var5, int var6) {
        this.nome = var1;
        this.estatisticas = new Estatisticas(var2, var3, var4);
        this.sequenciaVitoriasAtual = var5;
        this.melhorSequencia = var6;
    }
    public String getNome() {
        return this.nome;
    }
    public Estatisticas getEstatisticas() {
        return this.estatisticas;
    }
```

**`Estatisticas.java`:** Esta classe é uma parte da classe ***Jogador*** e é bem simples. Ela apenas armazena e atualiza os contadores de jogos, vitórias e derrotas do jogador. Ela é usada para manter um registro do desempenho do jogador.

Java

```
public class Estatisticas {
    private int jogos;
    private int vitorias;
    private int derrotas;

    public Estatisticas() {
        this.jogos = 0;
        this.vitorias = 0;
        this.derrotas = 0;
    }

    public Estatisticas(int var1, int var2, int var3) {
        this.jogos = var1;
        this.vitorias = var2;
```

```
        this.derrotas = var3;
    }

    public void registrarVitoria() {
        ++this.jogos;
        ++this.vitorias;
    }
```

## Descrição do Sistema

### Principais Funcionalidade

O sistema é composto por uma interface gráfica simples, intuitiva e funcional, que se adapta às mecânicas do jogo. As principais funcionalidades são:

- **Interface Principal:** Exibe informações como o número de jogos, partidas ganhas e perdidas. Também mostra as informações da partida atual, como tentativas restantes e letras já utilizadas.
- **Controle do Jogo:** Botões de interface gráfica permitem ao jogador iniciar uma nova partida, que interrompe a partida atual se houver uma em andamento. Há também um botão para sair do jogo, que exibe as estatísticas finais antes de fechar o programa.
- **Entrada de Palpite:** O jogador utiliza um campo de texto para inserir uma palavra de 5 letras. O sistema valida a entrada, aceitando apenas palavras de 5 letras que existam no banco de palavras.
- **Representação de Tentativas:** A interface gráfica representa cada tentativa de palavra por meio de quadrados coloridos, que indicam o estado de cada letra.
- **Banco de Palavras:** O jogo lê um arquivo de texto que contém palavras de 5 letras, que servem como o banco de palavras secretas.

### Entidades Manipuladas

- **`Jogo`**: Gerencia a lógica de uma partida, incluindo a palavra secreta e as tentativas restantes.
- **`Bancopalavras`**: Responsável por ler e carregar o banco de palavras na memória, selecionar aleatoriamente a próxima palavra secreta e garantir que palavras já usadas não sejam repetidas na mesma rodada.
- **`Estatisticas`**: Armazena e gerencia os dados de jogos, vitórias e derrotas do jogador, com capacidade de salvar e carregar esses dados de um arquivo de texto.
- **`Interface`**: A classe principal que lida com a interface gráfica, coordena a interação entre o jogador e as demais entidades, e atualiza a exibição do jogo.

### Interação com o Usuário

A interação com o usuário é totalmente mediada pela interface gráfica. O jogador insere a palavra no campo de texto e clica no botão "Enviar". As informações de progresso e estatísticas são exibidas em

**`JLabels`**, e as validações de entrada e mensagens de fim de jogo são apresentadas em caixas de diálogo (**`JOptionPane`**).

## Decisões de Projeto

### Modelagem

A modelagem do sistema foi baseada na criação de classes coesas e com responsabilidades bem definidas. A classe **`Main`** é o ponto de entrada, a **`Interface`** é o "cabeça" do jogo, e as classes **`Jogo`**, **`Bancopalavras`** e **`Estatisticas`** lidam com as lógicas específicas.

**Herança e Polimorfismo:** a herança é utilizada na classe interna **`LimiteCampoTexto`**, que estende **`javax.swing.text.PlainDocument`** para limitar o número de caracteres em um campo de texto. O polimorfismo também é utilizado através de interfaces como **`ActionListener`** e **`KeyListener`**, onde a **`Interface`** implementa os métodos de escuta de eventos para os botões e o campo de texto.

**Encapsulamento:** Todos os atributos das classes de domínio (**`Jogo`, `Bancopalavras`, `Estatisticas`**) são privados, e o acesso a eles é feito através de métodos públicos (**`getters`** e **`setters`**), garantindo o controle sobre a modificação do estado do objeto. Por exemplo, as estatísticas são modificadas apenas através dos métodos **`registrarVitoria()`** e **`registrarDerrota()`**.

### Estruturas da Biblioteca Swing

**`JFrame`:** A janela principal do jogo.

**`JPanel`:** Usados para organizar os componentes em painéis, utilizando **`BorderLayout`**, **`GridLayout`** e **`FlowLayout`** à disposição.

**`JLabel`, `JButton`, `JTextField`:** Componentes padrão para a exibição de texto, criação de botões e entrada de dados.

**`JOptionPane`:** Utilizado para exibir mensagens de diálogo, como avisos e resultados da partida.

### Persistência de Dados

As estatísticas do jogador (**`jogos`**, **`vitórias`**, **`derrotas`**) são salvas em um arquivo de texto (**`termo-estatisticas.txt`**).

O formato do arquivo é CSV (Comma-Separated Values), onde os dados são separados por vírgulas.

A classe **`Estatisticas`** é responsável pela lógica de leitura e escrita. O carregamento ocorre quando a classe é instanciada, lendo a primeira linha do arquivo. O salvamento é realizado sempre que uma vitória ou derrota é registrada, garantindo que os dados estejam sempre atualizados.

## Arquitetura do Projeto

### Descrição dos Arquivos

**`Main.java`:** Ponto de entrada do programa. Responsável por verificar os argumentos da linha de comando e iniciar a interface gráfica.

**`Interface.java`**: A classe principal da interface gráfica. Ela cria a janela, os painéis e os componentes. Agrega as outras classes (**`Jogo`**, **`Bancopalavras`**, **`Estatisticas`**) e coordena o fluxo do jogo.

**`Jogo.java`:** Contém a lógica de uma única partida. Gerencia a palavra secreta, as tentativas e a avaliação dos palpites.

**`Bancopalavras.java`:** Manipula o arquivo de texto com o banco de palavras. Carrega as palavras na memória, escolhe a palavra secreta para cada jogo e garante que as palavras não se repitam.

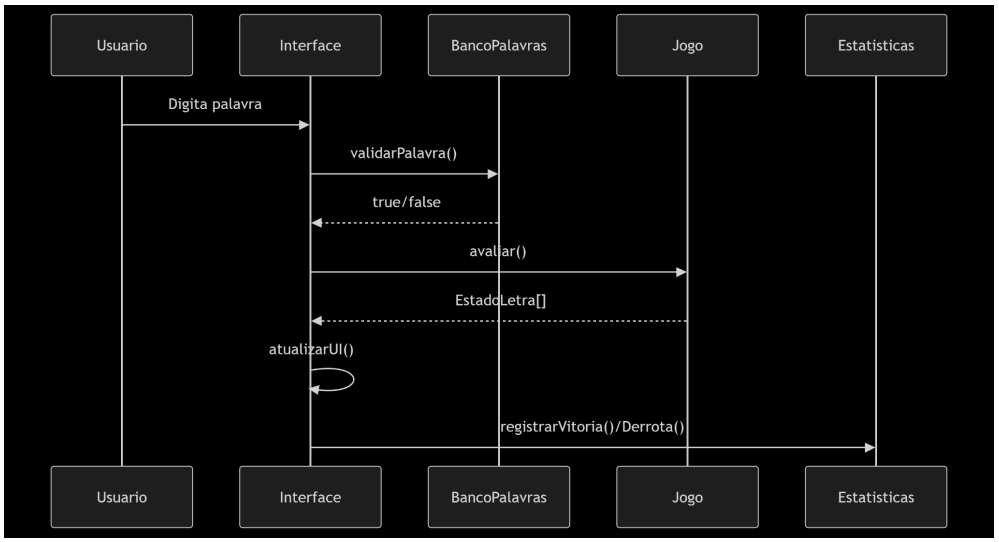
**`Estatisticas.java`:** Controla as estatísticas do jogador. Carrega e salva os dados de progresso em um arquivo para persistência.

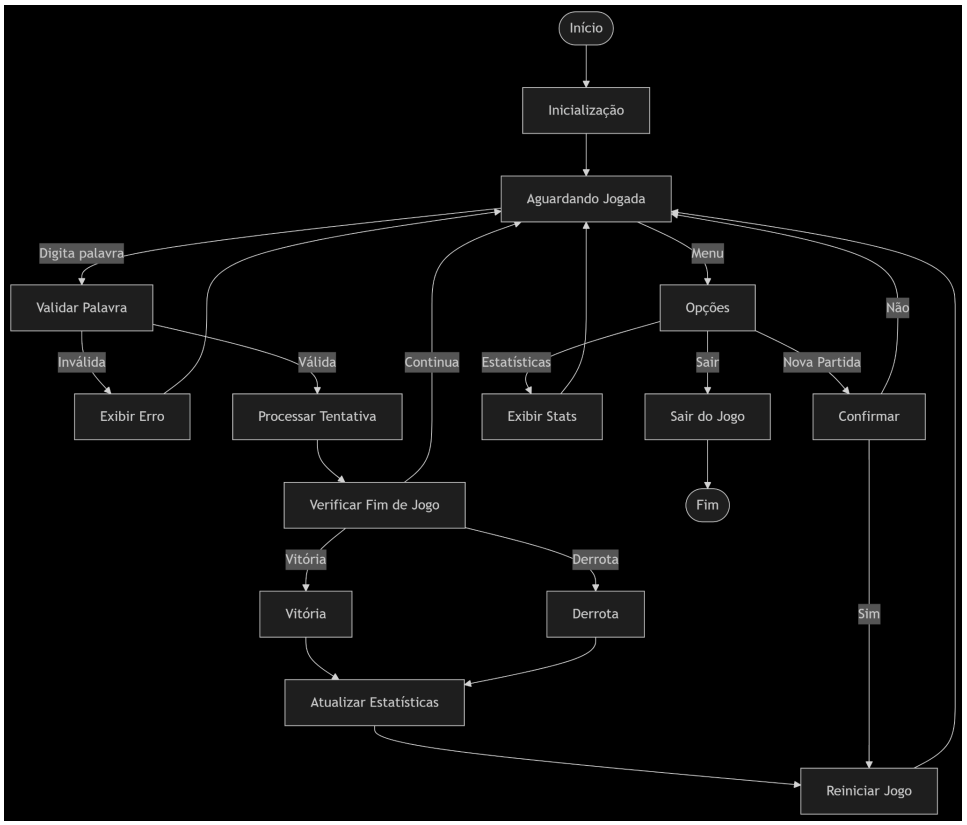
Funcionamento do **`main`:** O método **`main`** na classe **`Main`** é o ponto de partida do programa. Ele verifica se o usuário forneceu o caminho para o banco de palavras como um argumento na linha de comando. Se o argumento estiver ausente, ele exibe uma mensagem de erro e encerra o programa. Em seguida, ele utiliza **`SwingUtilities.invokeLater`** para criar a interface gráfica de forma segura na thread de eventos do Swing, garantindo que a aplicação se comporte corretamente.

### Relacionamento entre Classes

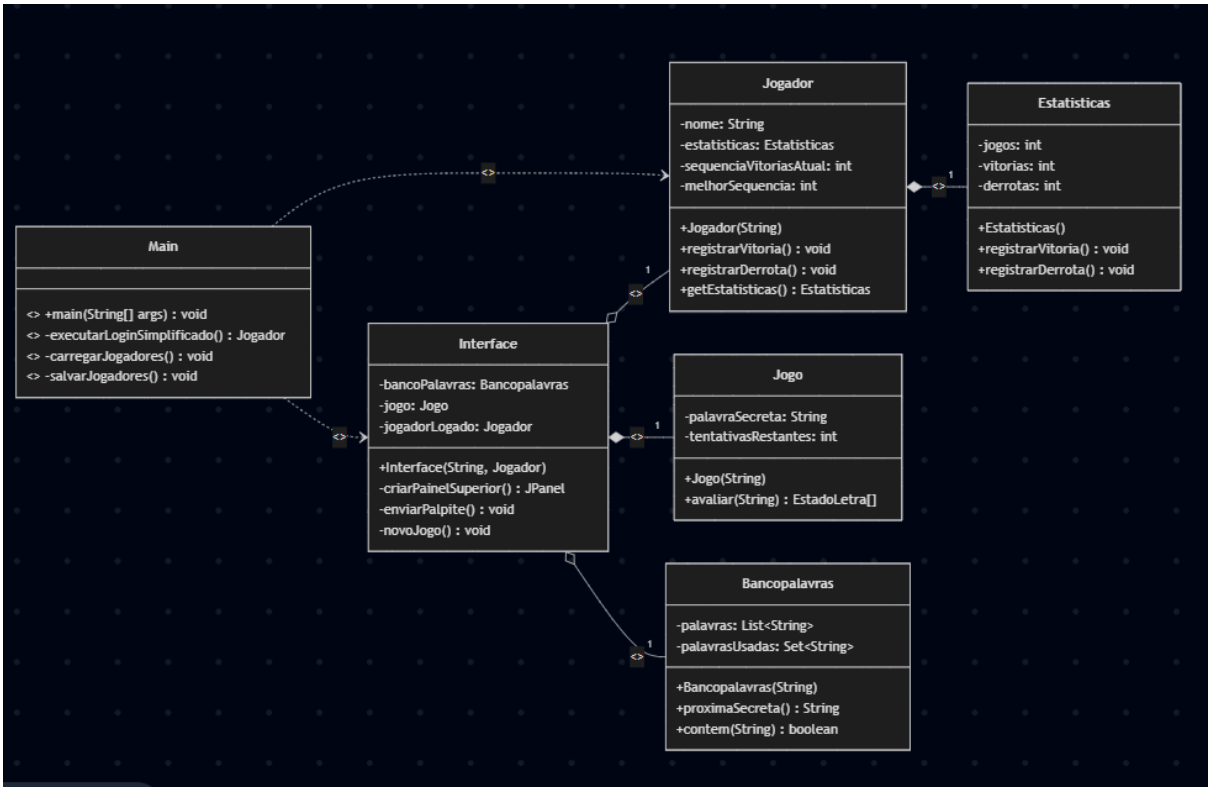
A classe **Interface** associa-se a uma instância de Jogo, uma de **Bancopalavras** e uma de **Estatisticas**. A classe **Jogo** não se associa diretamente a outras classes de domínio; a lógica de avaliação é contida nela, e os resultados são utilizados pela **Interface.** A classe **Estatisticas** se associa a um arquivo para persistência, mas não a outras classes de domínio.

## Diagramas de Fluxo



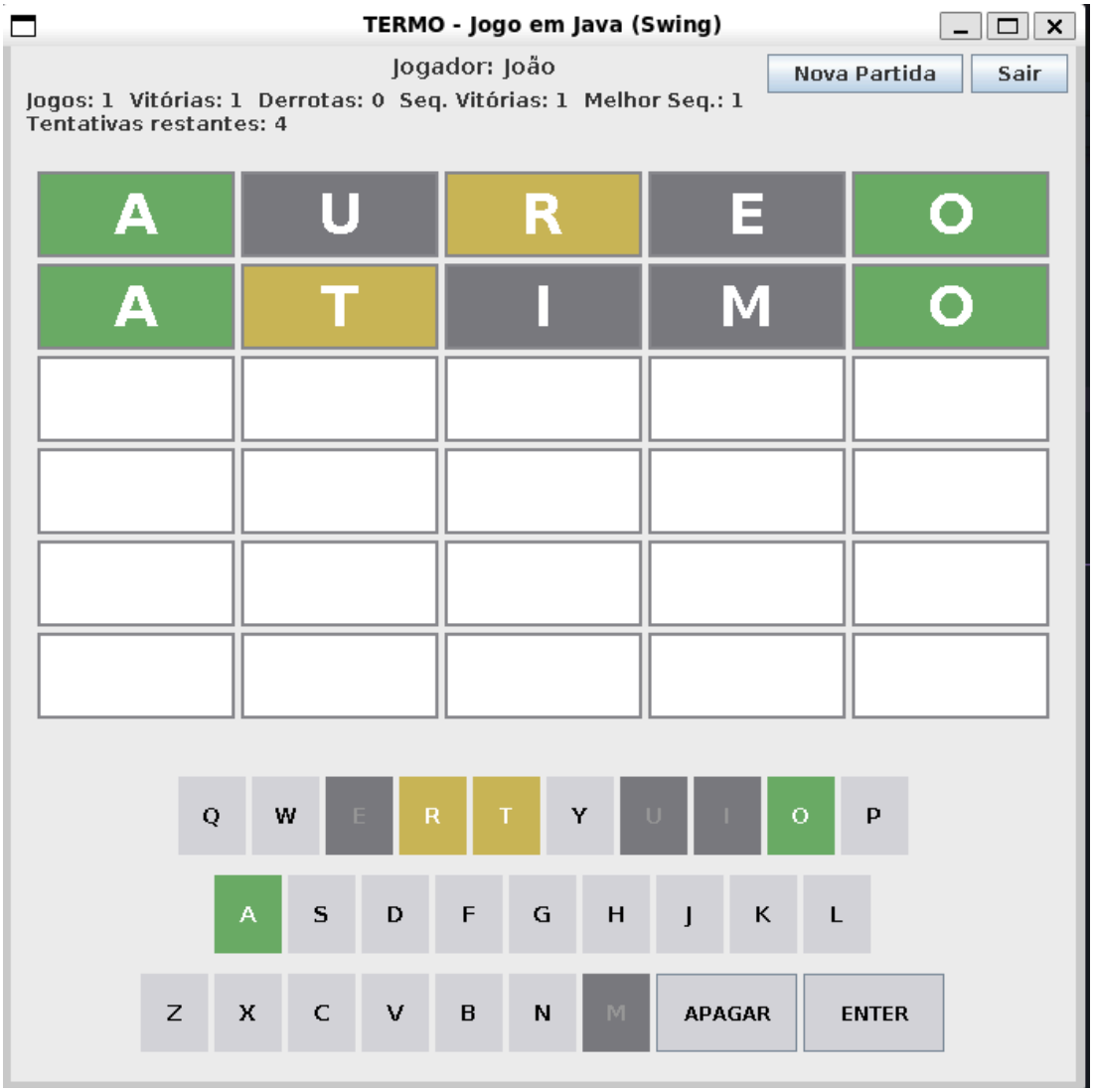



Diagramas UML

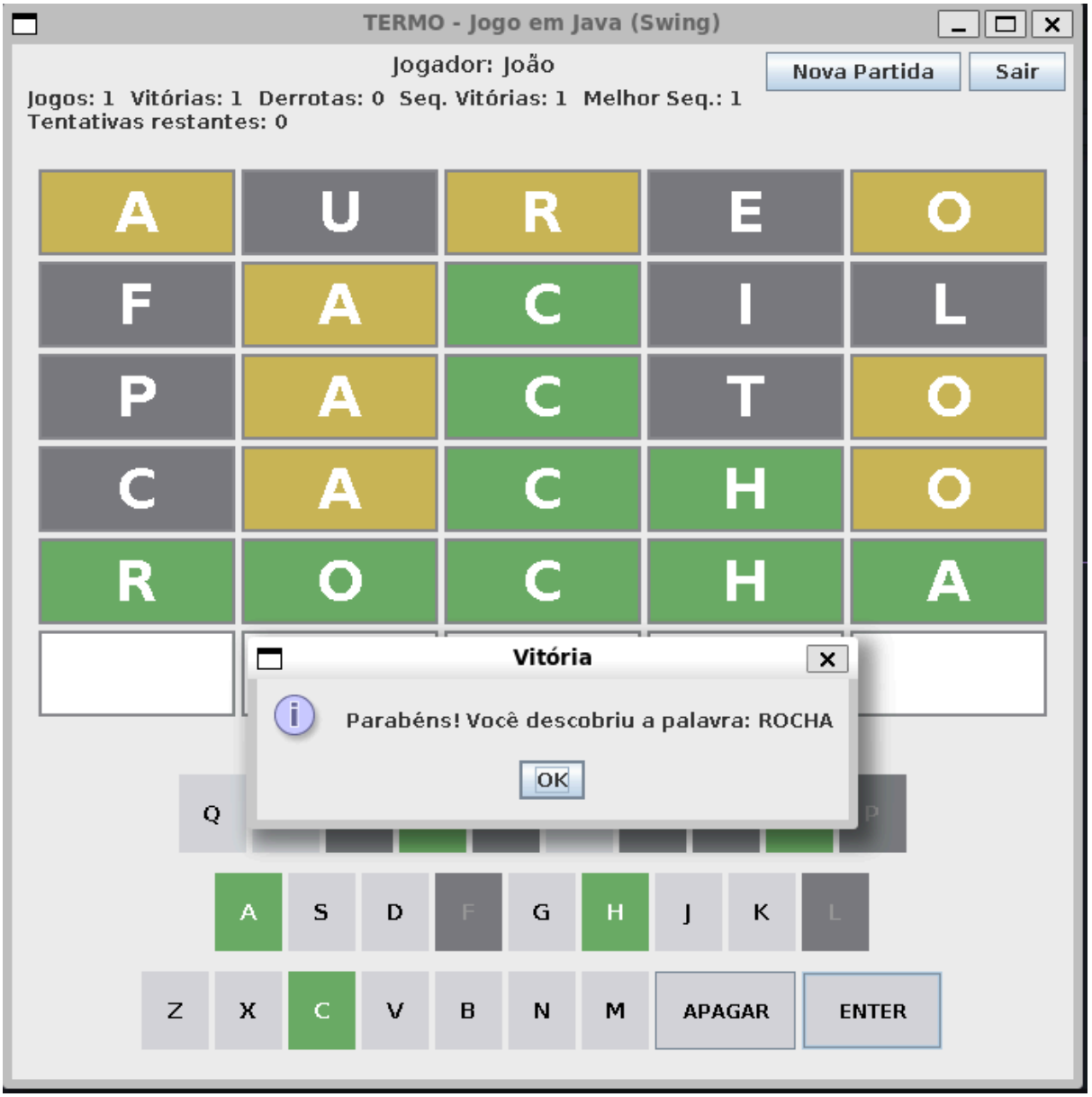


## Testes Realizados

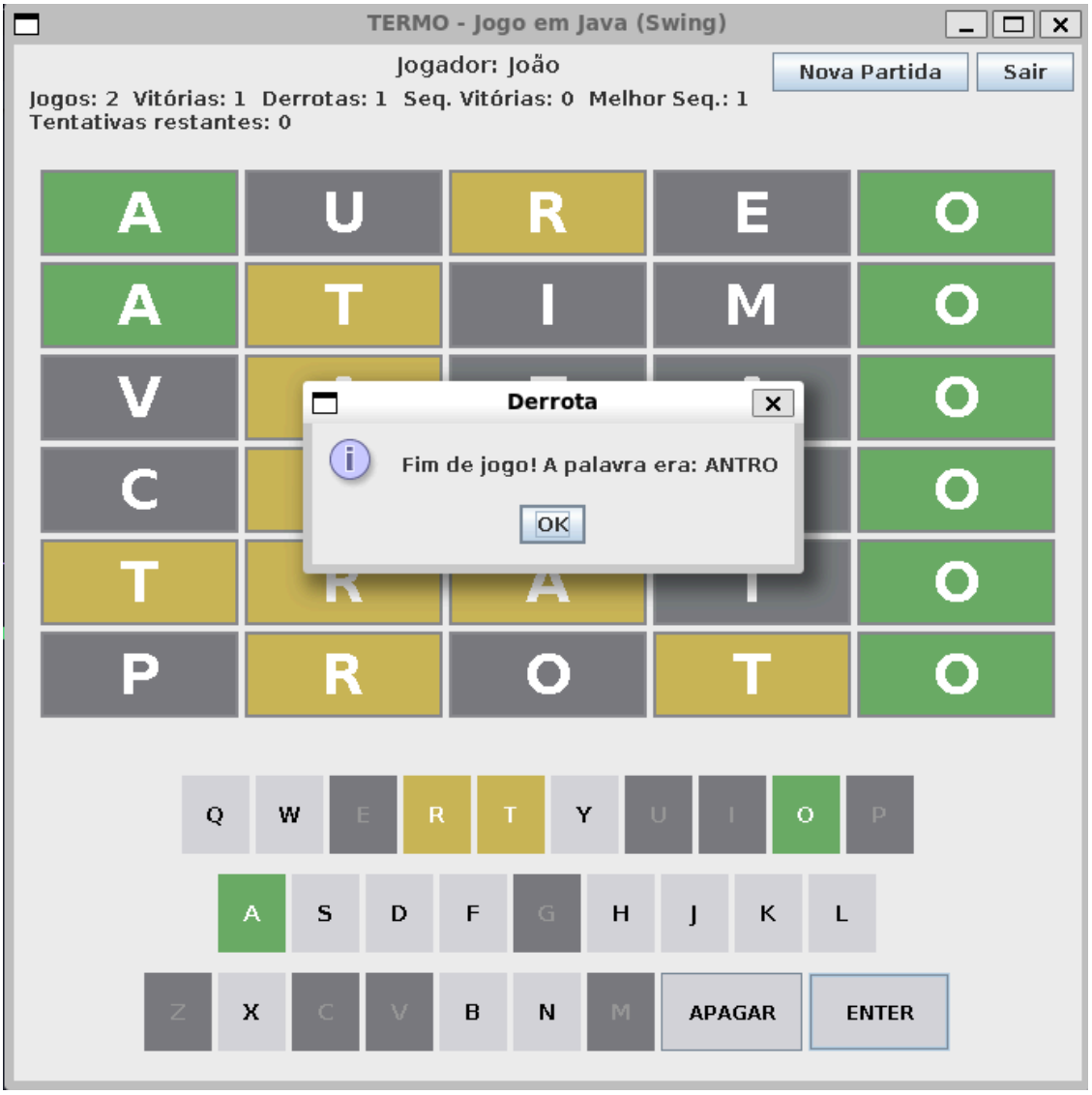
**Cenário 1 (Fluxo Principal):** Testamos o fluxo completo de uma partida, do início ao fim, incluindo palpites corretos, presentes e ausentes, e verificou-se se as cores dos quadrados e o número de tentativas restantes eram atualizados corretamente.



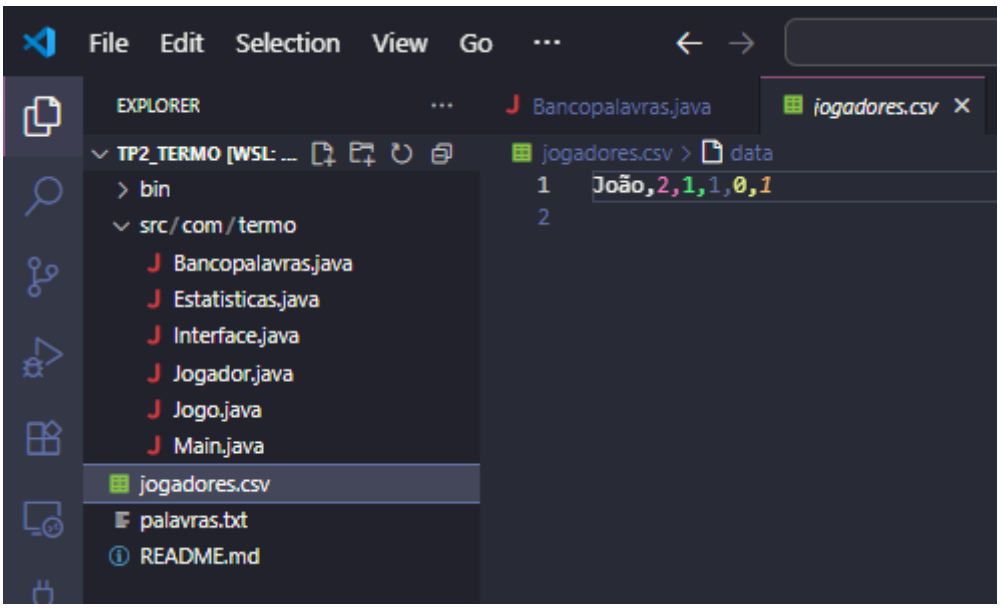
**Cenário 2 (Vitória):** Um palpite foi inserido para a palavra secreta. O sistema deve exibir uma mensagem de vitória, registrar a vitória nas estatísticas e reiniciar o jogo.



**Cenário 3 (Derrota):** Foram feitas 6 tentativas incorretas. O sistema deve exibir uma mensagem de derrota, revelar a palavra secreta, registrar a derrota nas estatísticas e iniciar um novo jogo.



**Cenário 4 (Persistência):** Após vitórias e derrotas, o programa foi fechado e reaberto para verificar se as estatísticas foram carregadas corretamente do arquivo **`jogadores.csv`**




## Tratamento de Erros

- O programa trata entradas inválidas de várias maneiras. Se o usuário tentar inserir uma palavra com menos de 5 letras, uma mensagem de aviso é exibida. Se a palavra não estiver no banco de palavras, outra mensagem é exibida.
- O programa também trata o erro de inicialização se o caminho para o banco de palavras não for fornecido como argumento da linha de comando, exibindo uma mensagem de uso e encerrando o programa.

Implementação do ponto extra

Foi implementado o sistema de Usuários, permitindo que jogos distintos sejam criados para usuários distintos, sendo possível salvar seu progresso ou recuperá-lo. A classe adicionada foi a **<Jogador.java>**

Java

```java
package com.termo;

public class Jogador {
    private final String nome;
    private final Estatisticas estatisticas;
    private int sequenciaVitoriasAtual;
    private int melhorSequencia;

    public Jogador(String nome) {
        this.nome = nome;
        this.estatisticas = new Estatisticas();
        this.sequenciaVitoriasAtual = 0;
        this.melhorSequencia = 0;
    }

    public Jogador(String nome, int jogos, int vitorias,
int derrotas, int seqAtual, int melhorSeq) {
        this.nome = nome;
        this.estatisticas = new Estatisticas(jogos,
vitorias, derrotas);
        this.sequenciaVitoriasAtual = seqAtual;
        this.melhorSequencia = melhorSeq;
```

## Conclusão

A implementação do jogo TERMO demonstrou a aplicação bem-sucedida de conceitos de POO em um projeto prático. O design modular, com classes especializadas e bem definidas (**`Jogo`**, **`Bancopalavras`**, **`Estatisticas e Jogador`**), resultou em um código coeso e de fácil manutenção. O uso da biblioteca Swing permitiu a criação de uma interface gráfica funcional e intuitiva. A persistência de dados foi implementada para as estatísticas, garantindo que o progresso do jogador seja mantido.

**Dificuldades Encontradas:** Uma das principais dificuldades foi a coordenação entre a lógica de jogo (em **`Jogo.java`**) e a interface gráfica (em **`Interface.java`**). A lógica de avaliação dos palpites, especialmente a contagem de letras presentes para evitar duplicação em letras ausentes, também exigiu uma abordagem cuidadosa. Todo o processo de validação das letras e uso do banco de palavras demandou atenção especial para que o código funcionasse como pedido.

No mais, foi um trabalho bem pensado e interessante de ser feito, uma vez que permitiu maior domínio dos conceitos aprendidos em aula e dos integrantes do grupo desenvolverem maior familiaridade com a linguagem Java.