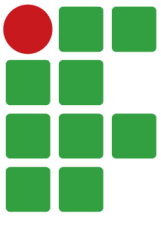


INSTITUTO FEDERAL
Minas Gerais
Campus Ouro Branco

2º ANO DO CURSO INTEGRADO EM INFORMÁTICA

HENRIQUE LEÃO & JOÃO PEDRO DE CASTRO

DOCUMENTAÇÃO DO TRABALHO PRÁTICO 3 - JOGO DA FORCA

Ouro Branco - MG

2023

INTRODUÇÃO

Descrição:

O código no qual essa documentação se refere foi desenvolvido para o ilustríssimo Seu José, que nos pediu com todo o carinho para fazermos uma versão digital do Jogo da Forca, a fim de instigar seus netos (que há muito não se importam com jogos do tipo) a jogarem esse clássico com ele.

Como já estamos treinados na arte da programação, não pensamos duas vezes antes de começar a desenvolver o projeto, que aplica com primor diversos conceitos básicos e utiliza muitos componentes da biblioteca Swing.

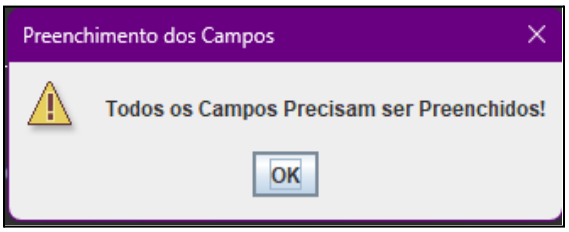
Segue o seu funcionamento:

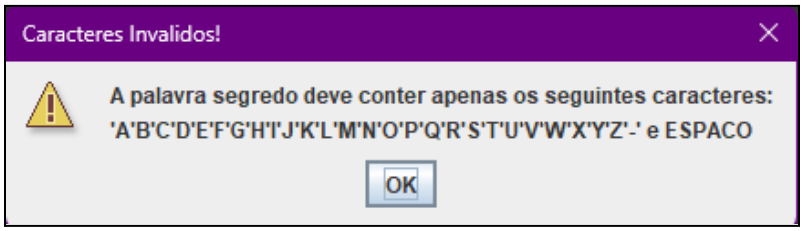
Funcionamento:

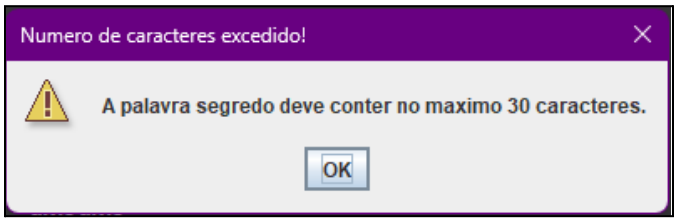
Ao executar o programa, nos deparamos com a seguinte tela:

Para iniciar o jogo, é necessário que o Jogador 1 informe uma “Palavra Segredo” válida no primeiro campo e a “Dica” no segundo campo, apertando o botão “COMEÇAR!” em seguida.

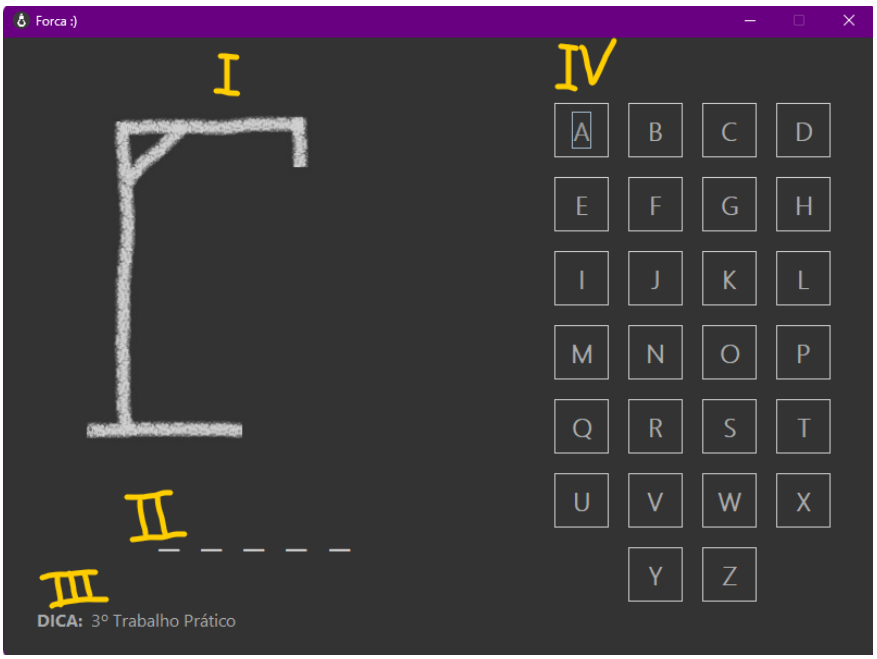
Após isso, algumas verificações são feitas e, se caso o Jogador 1 deixou algum campo vazio, utilizou caracteres inválidos ou ultrapassou o máximo de caracteres permitidos, os seguintes avisos aparecem (impedindo que a partida se inicie):





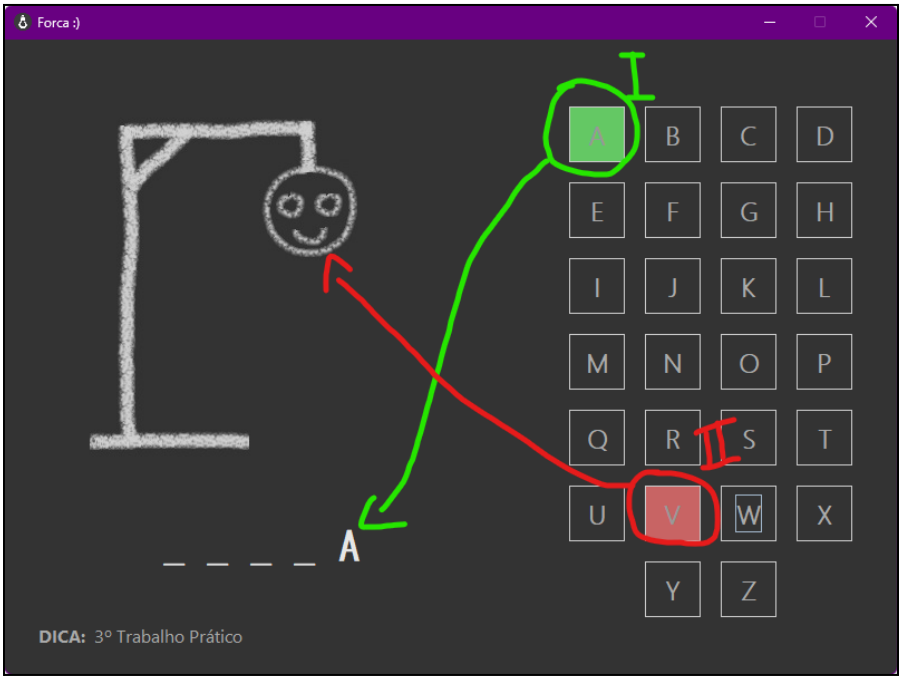


Se não houve nenhum problema com as verificações, o jogo se inicia:



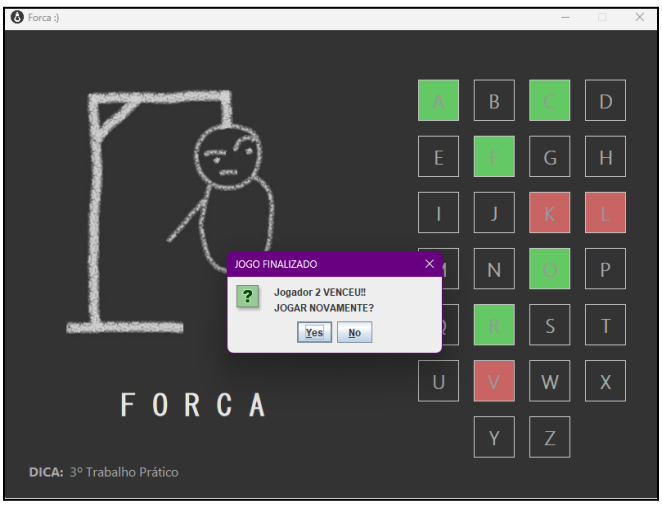
Em **I** temos uma label com a imagem da forca, que muda conforme os erros do P2.
Em **II** está a label que representa a palavra escolhida pelo P1, que atualiza quando há acertos.
Em **III** está a label com a dica que o P1 informou.
E em **IV** estão os botões, formando o teclado por onde o P1 escolhe as letras.

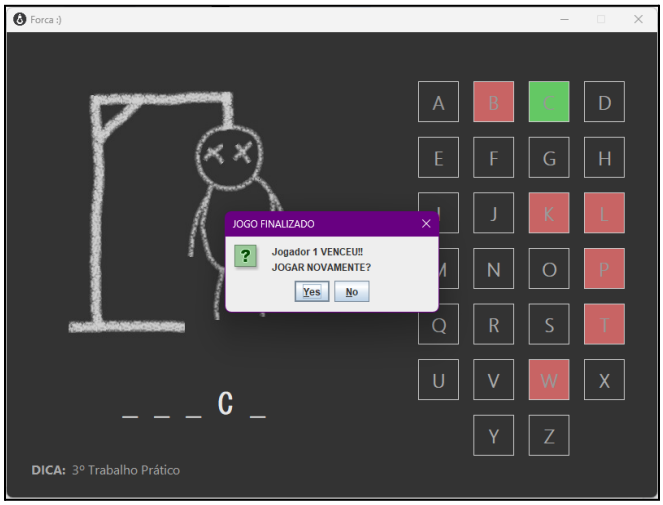
Assim, temos 2 cenários possíveis:



I - O P2 escolhe uma letra contida na “palavra segredo”, fazendo com que o botão fique verde e mostrando em qual posição (ou quais posições) da palavra se encontra a letra escolhida.
II - O P2 escolhe uma letra que não está contida na “palavra segredo”, deixando o botão vermelho e atualizando a label da imagem da forca.
Em ambos os casos os botões são desativados após serem apertados.

Para que a partida finalize, é necessário que (1) o P2 acerte todas as letras da “palavra segredo”, onde ele é o vencedor ou que (2) o P2 escolha 6 letras que não estão presentes na palavra, onde o P1 é o vencedor.



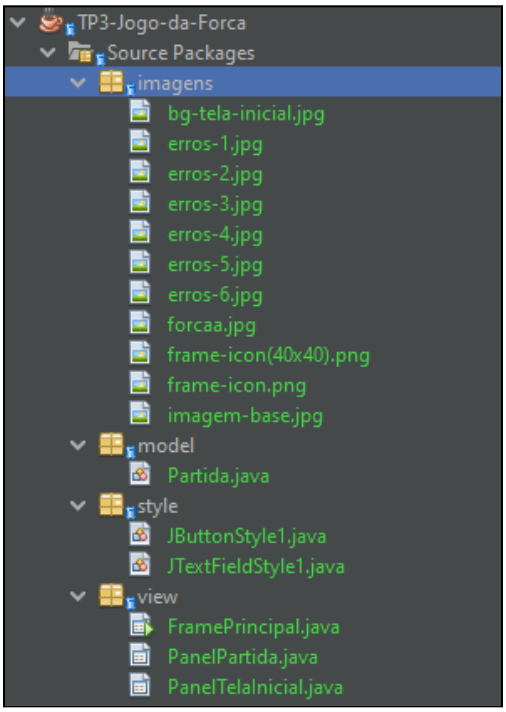


É importante mencionar que o tamanho da fonte da label da palavra se adapta a quantidade de caracteres presentes na “palavra segredo”:

DESENVOLVIMENTO

Organização:

Não há nada como um projeto bem estruturado e organizado.
Logo, vamos introduzi-lo pela maneira com que organizamos os arquivos em diferentes pacotes, segundo suas funções:



imagens: nesse pacote estão todas as imagens utilizadas no projeto, desde os ícones às imagens do boneco.

model: contém a classe de modelo da partida, onde estão declarados os atributos e métodos responsáveis por representar uma partida do jogo.

style: aqui estão as classes de customização de componentes, que utilizamos para facilitar o desenvolvimento.

view: inclui todas as classes responsáveis pela interface gráfica e interações com o usuário.

Classes:

Agora, podemos nos aprofundar no código presente em cada classe.

model > Partida

Como dito anteriormente, essa classe é responsável por representar uma partida, contendo as seguintes propriedades:

Atributos:

- **- palavraSegredo: String >>** armazena a palavra segredo que o P1 digitou;
- **- dica: String >>** armazena a dica;
- **- erros: int >>** contém o números de letras não existentes que o P2 escolheu.
- **- vencedor: int >>** armazena o valor “0” caso ainda não haja um vencedor e “1” ou “2” (representando o jogador) caso haja.
- **- palavraMostrada: char[] >>** representa a palavra que o P2 tem acesso ao jogar pela GUI.

Contrutores:

```
public Partida() {
}

public Partida(String palavraSegredo, String dica) {
    this.palavraSegredo = palavraSegredo;
    this.dica = dica;
    this.erros = 0; // ao iniciar a partida, na ha erros
    this.vencedor = 0; // vencedor = 0 significa que ainda nao ha um vencedor

    this.palavraMostrada = new char[palavraSegredo.length()];
    configurarPalavraMostrada();
}
```

Temos um construtor default e um construtor que recebe as Strings **palavraSegredo** e **dica** como parâmetro. Ao ser criada, a partida não possui erros nem vencedor, por isso os atributos **erros** e **vencedor** são inicializados com o valor 0. O array estático **palavraMostrada** é instanciado utilizando o número de caracteres da **palavraSegredo** como a quantidade de índices. E por fim o método **configurarPalavra Mostrada()** é chamado.

Encapsulamento:

```
//<editor-fold defaultstate="collapsed" desc=" GETTERS AND SETTERS ">
public String getPalavaSegredo() {...3 lines }

public void setPalavaSegredo(String palavaSegredo) {...3 lines }

public String getDica() {...3 lines }

public void setDica(String dica) {...3 lines }

public int getErros() {...3 lines }

public int getVencedor() {...3 lines }

public void setVencedor(int vencedor) {...3 lines }

public char[] getPalavraMostrada() {...3 lines }
//</editor-fold>
```

Métodos:

- **- configurarPalavraMostrada(): void**
  Método responsável por configurar o array **palavraMostrada** de acordo com a **palavraSegredo**, substituindo por **‘_’** (underline) todos os caracteres que não são um **'-'** (hífen) ou **' '** (espaço).

```
private void configurarPalavraMostrada(){
    DESCRICAO */

    for(int i = 0; i < palavraMostrada.length; i++){
        char c = palavraSegredo.charAt(i);
        if(c != ' ' && c != '-'){
            palavraMostrada[i] = '_';
        } else {
            palavraMostrada[i] = c;
        }
    }
}
```

- **+ getPalavraMostradaComoStringFormatada(): String**
  Método que retorna uma String que concatena todos os caracteres da array **palavraMostrada**, dando um espaço entre eles.

```
public String getPalavraMostradaComoStringFormatada() {
    DESCRICAO */

    String str = "";
    for(int i = 0; i < palavraMostrada.length; i++){
        str += palavraMostrada[i] + " ";
    }
    return str;
}
```

- **+ validarLetra(String): Color**
  Verifica se a **palavraSegredo** contém a letra que o P2 escolheu (atualizando a **palavraMostrada** e retornando a cor verde) ou se não contém (incrementando 1 erro e retornando a cor vermelha).

```
public Color validarLetra(String letra){
    DESCRICAO */

    Color cor;
    if(palavraSegredo.contains(letra)){   // jogador 2 ACERTA uma letra
        atualizarPalavraMostrada(letra.charAt(0));
        cor = new Color(100,200,100); // cor = VERDE
    }else{ // jogador 2 ERRA uma letra
        erros++;
        cor = new Color(200,100,100); // cor = VERMELHO
    }
    verificarSeHouveVencedor();
    return cor;
}
```

- **- atualizarPalavraMostrada(char): void**
  Percorre todos os caracteres da **palavraSegredo**, verificando em qual (ou quais) índice(s) a **letra** recebida aparece. Então, na **palavraMostrada**, substitui o **'_'** pela **letra** no(s) mesmo(s) índice(s).

```
private void atualizarPalavraMostrada(char letra){
    /* DESCRICAO */

    for(int i = 0; i < palavraSegredo.length(); i++){
        if(palavraSegredo.charAt(i) == letra){
            palavraMostrada[i] = letra;
        }
    }
}
```

- **- verificarSeHouveVencedor(): void**
  Verifica se houve mais de 6 erros, atribuindo o valor 1 ao atributo **vencedor** ou se a **palavra Mostrada** não contém **'_'**, atribuindo o valor 2.

```
private void verificarSeHouveVencedor(){
    /* DESCRICAO */

    if(this.erros >= 6) {
        // se o jogador 2 errou 6 vezes, o jogador 1 vence
        this.vencedor = 1;
    }else if(!Arrays.toString(this.palavraMostrada).contains("_")){
        // se a palavra foi descoberta (nao contem mais "_"), o jogador 2 vence
        this.vencedor = 2;
    }
}
```

view > FramePrincipal (extends **JFrame)**

Essa classe representa o único frame da nossa aplicação e onde o método “main” está presente.

Atributos:

- **- conteudoFrame: Container >>** objeto no qual será feito a troca de painéis e apresenta o conteúdo atual do frame;

Construtor:

```
public FramePrincipal() {
    initComponents();

    this.setLayout(new BorderLayout()); // -> configurando o layout do frame

    // setando o icone do frame
    this.setIconImage(
            new ImageIcon(getClass().getResource("/imagens/frame-icon.png")).getImage());

    conteudoFrame = this.getContentPane();

    // o painel PanelTelaInicial e o primeiro a ser exibido quando o codigo e executado
    trocarPainel(new PanelTelaInicial(this));
}
```

Ao instanciar um **FramePrincipal**, chamamos o método **initComponents()**, que faz toda a parte de configuração dos componentes. Depois, configuramos o layout do frame como **BorderLayout** e “setamos” a imagem **frame-icon.png** como o ícone do frame. Atribuímos o container retornado pelo método **getContentPane()** ao atributo **conteudoFrame** e chamamos o método **trocarPainel(Jpanel),** passando um novo **PanelTelaInicial** como parâmetro, já que esse é o primeiro painel a ser apresentado para o usuário.

Métodos:

- **+ main(String[]): void**
  Método que inicia o programa, configura o look and feel e instancia o **FramePrincipal.**

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    java.awt.EventQueue.invokeLater(() -> new FramePrincipal().setVisible(true));
}
```

- **+ trocarPainel(JPanel): void**
  Método que troca o conteúdo que está sendo exibido pelo frame para o painel recebido como parâmetro.

```
public void trocarPainel(JPanel painelNovo){
    DESCRICAO */

    conteudoFrame.removeAll(); // limpa o container

    conteudoFrame.add(painelNovo); // adiciona o painel novo ao container

    validate(); // atualiza a interface grafica
    setVisible(true); // deixa o painel visivel
    conteudoFrame.repaint(); // renderiza a tela
}
```

view > PanelTelaInicial (extends JPanel)

Primeira tela que aparece após a execução do código, onde o P1 preenche os campos de texto e onde ocorre a validação da **palavraSegredo**.

Atributos:

- **- framePai: FramePrincipal >>** referencia;
- **- CARACTERES_DISPONIVEIS: final String >>** constante que contém os caracteres válidos para a **palavraSegredo.**
- **- MAX_CARACTERES: final int >>** constante que contém o número máximo de caracteres permitidos para a **palavraSegredo.**
- **- btnComecarJogo: JButtonStyle1 >>** botão que começa o jogo ao ser pressionado
- **- labelIconForca: JLabel >>** label que mostra o ícone do jogo.
- **- txtFieldPalavraSegredo: JTextFieldStyle1 >>** text field que o P1 informa a "palavra segredo".
- **- txtFieldDica: JTextFieldStyle1 >>** text field que o P1 informa a dica.

Construtor:

```
public PanelTelaInicial(FramePrincipal framePai) {
    initComponents();

    this.framePai = framePai;

    this.labelImagemForca.setIcon(
            new ImageIcon(getClass().getResource("/imagens/frame-icon.png")));
}
```

Ao instanciar um **PanelTelaInicial,** precisamos passar um **FramePrincipal** como parâmetro. Chamamos o método **initComponents()** para inicializar e configurar os componentes novamente, definimos o atributo **framePai** como o **FramePrincipal** passado e "setamos" a imagem "**frame-icon.png**" como ícone da label **labelIconForca.**

Métodos:

- **- iniciarJogo(): void**
  Cria uma partida e troca a tela para o **PainelPartida** caso não haja problema com as verificações.

```
private void iniciarJogo(){
    DESCRICAO */

    String palavraSegredo = txtFieldPalavraSegredo.getText().toUpperCase();
    String dica = txtFieldDica.getText();


    if(!validarPalavraSegredoEDica(palavraSegredo, dica)) return;

    this.framePai.trocarPainel(new PanelPartida(framePai, new Partida(palavraSegredo, dica)));
}
```

- **- validarPalavraSegredoEDica(String, String): void**
  Realiza três verificações (se todos os campos estão preenchidos, se a **palavraSegredo** possui apenas caracteres válidos e se a **palavraSegredo** possui mais caracteres do que o permitido) nas Strings dos campos, retornando false e comunicando com o usuário caso haja algo de errado ou retornando true caso esteja tudo certo.

```
private boolean validarPalavraSegredoEDica(String palavraSegredo, String dica){
    DESCRICAO */

    boolean[] validacoes = {true, true, true};
    String mensagem = "", titulo = "";


    Primeira Validacao -> todos os campos estao preenchidos?

    Segunda Validacao -> a palavraSegredo possui apenas caracteres validos?

    Terceira Validacao -> a palavraSegredo possui mais caracteres que o maximo permitido?

    // verificando se alguma validacao falhou
    if(!validacoes[0] || !validacoes[1] || !validacoes[2]){
        JOptionPane.showMessageDialog(this,  mensagem, titulo, JOptionPane.WARNING_MESSAGE);
        return false;
    }

    return true;
}
```

Verificações:

```
if(palavraSegredo.isEmpty() || dica.isEmpty()){
    validacoes[0] = false;
    mensagem = "Todos os Campos Precisam ser Preenchidos!";
    titulo = "Preenchimento dos Campos";
}// </editor-fold>
```

```
for(int i = 0; i < palavraSegredo.length(); i++){
    if(!CARACTERES_DISPONIVEIS.contains(""+palavraSegredo.charAt(i))){
        validacoes[1] = false;
        mensagem = "A palavra segredo deve conter apenas os seguintes caracteres: \n" +
                CARACTERES_DISPONIVEIS;
        titulo = "Caracteres Invalidos!";
        break;
    }
}// </editor-fold>
```

```
if(palavraSegredo.length() > 30){
    validacoes[2] = false;
    mensagem = "A palavra segredo deve conter no maximo " + MAX_CARACTERES + " caracteres. ";
    titulo = "Numero de caracteres excedido!";
}// </editor-fold>
```

view > PanelPartida (extends JPanel)

Painel onde o jogo propriamente dito acontece.

Atributos:

- **- framePai: FramePrincipal >>** referencia o frame no qual o panel é atribuído;
- **- partida: Partida >>** referencia;
- **- erros: int >>** armazena a contagem de erros no panel;
- **- btnLetraA** à **btnLetraZ: JButtonStyle1 >>** botões que formam o teclado e chamam o método e têm seus eventos configurados para chamar o método **atualizarBotao(JButton)**.
- **- labelImagemForca: JLabel >>** label que mostra a imagem da forca de acordo com os erros.
- **- labelDica: JLabel >>** label que apresenta a dica para o P2.
- **- labelPalavra: JLabel >>** label que mostra a **palavraMostrada** para o usuário, que é modificada de acordo com os acertos.

Construtor:

```
public PanelPartida(FramePrincipal framePai, Partida partida) {
    initComponents();

    this.framePai = framePai;
    this.partida = partida;

    // configurando a primeira imagem da label "labelImagemForca"
    labelImagemForca.setIcon(new ImageIcon(getClass().getResource("/imagens/imagem-base.jpg")));

    if (partida.equals(null)) System.exit(0);
    labelDica.setText(partida.getDica()); // -> setando dica

    configurarFonteLabelPalavra(); // -> ajustando tamanho da fonte da labelPalavra

    atualizarUI(); // -> atualizando interface grafica

    this.erros = 0;
}
```

O construtor de um **PanelPartida** recebe um **FramePrincipal** e uma **Partida** como parâmetros. Novamente chamamos o método **initComponents()** e definimos os atributos **framePai** e **partida** como os objetos recebidos. Depois configuramos a imagem da label **labelImagemForca** como a imagem “**imagem-base.jpg**”, que é a primeira imagem que aparece ao iniciarmos a partida. Verificamos se a **partida** não está vazia (encerrando a execução caso esteja) e “setamos” a dica como texto da label **labelDica**. Então chamamos o método **configurarFonteLabelPalavra()** para ajustar o tamanho da **labelPalavra**, chamamos o método **atualizarUI()** para atualizar a interface gráfica e por fim definimos a contagem de **erros** do painel como 0.

Métodos:

- **- configurarFonteLabelPalavra(): void**
  Método responsável por adaptar o tamanho da fonte da **labelPalavra** de acordo com o número de caracteres da palavra, utilizando o cálculo (42-**numCaracteresPalavra**).

```
private void configurarFonteLabelPalavra() {
    /* DESCRICAO */

    int numCaracteresPalavra = partida.getPalavraMostrada().length;
    Font fonte = new Font("MS Gothic", 1, (42-numCaracteresPalavra));

    labelPalavra.setFont(fonte);
}
```

- **- atualizarUI(): void**
  Verifica o status da partida e atualiza a interface gráfica de acordo com os dados. Caso a contagem de **erros** do painel esteja desatualizada, trocamos a imagem da **labelImagemForca** e atualizamos a variável. Se o método **getVencedor()** do objeto **partida** retornar um número diferente de 0, significa que houve um vencedor, portanto chamamos o método **finalizarPartida()**.

```
private void atualizarUI() {
    /* DESCRICAO */

    // atualiza labelPalavra
    labelPalavra.setText(partida.getPalavraMostradaComoStringFormatada());

    // verifica se houve o jogador 2 errou a letra
    if(this.erros < partida.getErros()) {
        // se houve um erro, a imagem da forca e atualizada de acordo com o numero de erros
        labelImagemForca.setIcon(new ImageIcon(getClass().
                getResource("/imagens/erros-" + partida.getErros() + ".jpg")));
        this.erros = partida.getErros();
    }

    // se vencedor != 0, algum jogador venceu, portanto a partida e finalizada
    if(partida.getVencedor() != 0) finalizarPartida();

}
```

- **- atualizarBotao(JButton): void**
  Método chamado quando um botão é apertado, setando o retorno do método **validarLetra(String)** (VERMELHO -> não contém a letra e VERDE -> contém a letra) como cor de fundo do botão, passando a própria letra presente no texto do botão como parâmetro para verificação. Também muda a cor do texto do botão para preto, o desabilita e chama o método **atualizaUI()**.

```
private void atualizarBotao(JButton btn) {
    /* DESCRICAO */

    btn.setBackground(partida.validarLetra(btn.getText()));

    btn.setForeground(java.awt.Color.BLACK); // -> setando a cor da letra para preto
    btn.setEnabled(false); // -> botao e desabilitado ao ser apertado 1 vez
    atualizarUI(); // -> atualiza componentes graficos de acordo com as alteracoes da partida
}
```

- **- finalizarPartida(): void**
  Método responsável por terminar a partida caso haja algum vencedor, mostrando um **ConfirmDialog** para informar o vencedor e perguntar ao usuário se ele deseja jogar novamente (se o usuário apertar o botão “não”, o código para de ser executado, mas se “sim” foi a opção escolhida, trocamos para o painel **PanelTelaInicial** e o jogo recomeça).

```
private void finalizarPartida() {
    DESCRICAO */

    int opcao = JOptionPane.showConfirmDialog(this,
            "Jogador "+ partida.getVencedor() +" VENCEU!!\nJOGAR NOVAMENTE?",
            "JOGO FINALIZADO", JOptionPane.YES_NO_OPTION);

    // se o usuario escolher "nao", a execucao do codigo e parada
    if (opcao == JOptionPane.NO_OPTION) {
        System.exit(0);
        return;
    }
    framePai.trocarPainel(new PanelTelaInicial(framePai));
}
```

```
//<editor-fold defaultstate="collapsed" desc=" EVENTOS DOS BOTOES DO ALFABETO ">

/* ------------------------------------------------------------
    Aqui estao todos os eventos que foram configurados para
    cada botao do alfabeto presente no panel da partida.
    Quando um botao e apertado, ele se passa como parametro
    para o metodo "atualizarBotao", que realiza as demais acoes.
------------------------------------------------------------*/

private void btnLetraAActionPerformed(java.awt.event.ActionEvent evt) {
    atualizarBotao(btnLetraA);
}

private void btnLetraBActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraCActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraDActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraMActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraFActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraGActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraJActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraVActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraRActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraNActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraKActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btnLetraLActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }
```

CONCLUSÃO

Durante a realização do trabalho, podemos perceber que evoluímos muito na forma como programamos utilizando GUI (Graphical User Interface) e em como organizamos o código como um todo, com nomeações mais sugestivas de variáveis e métodos e separação das classes em diferentes pacotes.

Descobrimos diversas coisas úteis, como o **editor-fold** que nos permite deixar o código ainda mais organizado e limpo para leitura e a utilizar elementos customizados na criação da interface gráfica.

Tivemos muitos momentos em que quebramos a cabeça com perguntas do tipo: “como é que deixa a label menor se a palavra for muito grande? ”, que são partes fundamentais na resolução eficiente de problemas.

No geral, ficamos bem satisfeitos com o resultado final, mas não desconsideramos a necessidade de algumas otimizações aqui e ali.