UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01120 – TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS

Trabalho Final - Etapa 2

Bruno Castanho, Leandra Machado, Lucas Gomes, Maria Eduarda Casali Ricardo, Vitor Arguilar

UFRGS, 2025

Mudanças em Relação à Etapa 1

Não houve mudanças em relação aos detalhes referentes ao funcionamento do jogo especificados na etapa 1, mas durante a implementação foram identificadas lacunas referentes às classes previamente escolhidas para serem construídas. Dessa forma, em comparação à etapa 1 do trabalho, novas classes foram adicionadas e essas serão detalhadas abaixo.

Package UI

Public Class ComponentesUI

Métodos:

+ static botaoReiniciar(aoReiniciar: Runnable): JButton
+ static botaoSair( ): JButton
+ static painelComReiniciarESair(aoReiniciar: Runnable): JButton

Public Class Estilos

Atributos:

+ static AMARELO: Color
+ static CINZA: Color
+ static VERDE: Color
+ static AZUL: Color
+ static VERMELHO: Color
+ static FONTE_PADRAO: Font
+ static TAMANHO_TELA_JOGO: Dimension
+ static TAMANHO_TELA_PADRAO: Dimension

Package Telas

Public Class TelaRegras

Métodos:

+ static criar(cardLayout: CardLayout, painelCartoes: JPanel): JPanel

Public Class TelaJogo extends JPanel

Atributos:

- jogo: Jogo
- labelImagem: JLabel
- wordPanel: JPanel

- letterLabels[ ]: JLabel
- caixaJogo: JPanel
- cardLayout: CardLayout
- painelCartoes: JPanel
- painelLetrasTestadas: JPanel
- letrasTestadasLabel: JLabel
- labelCategoriaDica: JLabel

Métodos:
+ getPalavraAtual( ): Palavra
+ setupKeyBindings( ): void
- generate_image( ): Image
- setupWordDisplay(palavra: String): void
- revealLetter(guessedLetter: char, palavra: String): void
+ todasLetrasAcertadas(palavra: String): boolean
- atualizarLetrasTestadas( ): void
- resetGame( ): void

Construtor:
+ TelaJogo(CardLayout cardLayout, JPanel painelCartoes)

Public Class TelaGanhou

Atributos:
+ palavraEncerrada: Palavra

Métodos:
+ static criar(cardLayout: CardLayout, painelCartoes: JPanel): JPanel
+ static setPalavraEncerrada(palavra: Palavra): void

Public Class TelaPerdeu

Atributos:
+ palavraEncerrada: Palavra

Métodos:
+ static criar(cardLayout: CardLayout, painelCartoes: JPanel): JPanel
+ static setPalavraEncerrada(palavra: Palavra): void

Implementação

Nesta seção, detalharemos a implementação de algumas classes relacionadas à parte lógica do jogo e de tipos abstratos de dados previstos no projeto original. Classes com ênfase na interface serão omitidas por não fazerem parte do escopo desta disciplina.

Package *app*

public abstract class Categoria

Atributos:

+ nomecategoria: String

Métodos:

+ abstract exibirCategoria( ): String
+ abstract exibirDica( ): String

Construtor:

+ Categoria(String var1)

A classe abstrata Categoria serve como base para classes contidas no domínio do pacote categorias, visto mais adiante. Ela contém um atributo do tipo String, nomecategoria e um construtor que atribui um valor a este atributo. Os dois métodos abstratos não possuem corpo de código, pois são implementados em suas classes filhas, no pacote categorias.

public class Jogo

Atributos:

- numvidas: int
- palavra: Palavra
- letrasTestadas: Set<Character>
- listaPalavras: List<Palavra>

Métodos:

+ getPalavra( ): Palavra
+ getNumVidas( ): int
+ setNumVidas(int numvidas): void
+ checaLetra(char letra): int
+ retiraVida( ): void
+ getLetrasTestadas( ): Set<Character>
+ getStringPalavra( ): String
+ final trocaPalavra( ): void
+ reset( ): void

Construtor:

+ Jogo( )

A classe pública Jogo compreende atributos e métodos atrelados à lógica de funcionamento do jogo. Ela contém atributos para contagem de vidas do jogador (*numvidas*), armazenar a palavra randomizada (*palavra*), mostrar ao jogador um conjunto de letras já usadas em palpites (*letrasTestadas*) e uma lista contendo o banco de palavras que poderão aparecer no jogo (*listaPalavras*).

Seus métodos incluem, além de getters e setters relacionados à palavra, letras testadas e número de vidas restantes, um método para a checagem do palpite do usuário (*checaLetra*), que retorna um inteiro. 0, caso a letra já tenha sido testada; 1, caso a palavra possua a letra testada, e -1, caso a letra não esteja presente na palavra. Neste último caso, as vidas do jogador são decrementadas em 1 (*retiraVida*). Em ambos os casos de presença e ausência da letra na palavra, a letra usada no palpite é adicionada ao conjunto de letras testadas (*letrasTestadas*.*add*(*letra*)).

public class Palavra

Atributos:

- palavra: String
- numLetras: int
- nomeCategoria: Categoria

Métodos:

+ static criaListaPalavras( ): List<Palavra>
+ static escolhePalavraSecreta(List<Palavra> lista): Palavra
+ getStringPalavra( ): String
+ getNumLetras( ): int
+ getCategoria( ): Categoria
+ contémLetra(char Letra): boolean

Construtor:

+ Palavra(String palavra, int numLetras, Categoria nomeCategoria)

A classe Palavra contém atributos e métodos necessários para a compreensão e categorização das palavras do banco no código da aplicação. Os atributos de uma palavra incluem a palavra propriamente dita (*palavra*), o número de letras dela (*numLetras*), e sua categoria (*nomeCategoria*). A geração da lista de palavras, que compreende todo o dicionário incluso no código-fonte em formato *txt*, fica por conta do método *criaListaPalavras*. Ele realiza o parsing de cada um dos arquivos *txt* na pasta resources, e, uma vez lida a palavra, ela é categorizada de acordo com o arquivo no qual estava presente.

O método estático *escolhePalavraSecreta*, por sua vez, randomiza um valor inteiro e retorna uma palavra contida na lista elaborada pelo método descrito anteriormente.

Por fim, o método *contemLetra* retorna um booleano, que informa se uma letra está presente ou não na palavra secreta. Ele recebe a letra usada no palpite e normaliza a palavra secreta, colocando todas suas letras em minúsculo e removendo acentuação e cedilhas, de forma a facilitar a comparação da entrada do usuário com as letras da palavra aleatória.

Package *categorias*

public abstract class Animais extends Categoria

Atributos:

- \+ (super) nomecategoria: String

Métodos:

- \+ exibircategoria ( ): void
- \+ abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):

- \+ Animais(String nomecategoria)

public abstract class Paises extends Categoria

Atributos:

- \+ (super) nomecategoria: String

Métodos:

- \+ exibircategoria ( ): void
- \+ abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):

- \+ Paises(String nomecategoria)

public abstract class Personagem extends Categoria

Atributos:

- \+ (super) nomecategoria: String

Métodos:

- \+ exibircategoria ( ): void
- \+ abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):

- \+ Personagem(String nomecategoria)

As classes abstratas *Animais*, *Paises* e *Personagem* herdam o atributo *nomecategoria* da classe *Categoria*, contida no pacote *app*. Além disso, os métodos *exibirDica( )* e *exibirCategoria( )* também são herdados, porém, somente o último é implementado nas três. A implementação deste método, antes abstrato, retorna o nome da categoria (*String nomecategoria*) da palavra aleatória durante o jogo. Ainda assim, vale lembrar que as três classes permanecem abstratas, e não podem ser instanciadas em objetos.

public class America extends Paises

Atributos:
+ (super) nomecategoria: String

Métodos:
+ exibirdica( ): void

Construtor:
+ America(String categoria)

public class Asia extends Paises

Atributos:
- \+ (super) nomecategoria: String

Métodos:
- \+ exibirdica( ): void

Construtor:
- \+ Asia(String nomecategoria)

public class Europa extends Paises

Atributos:
- \+ (super) nomecategoria: String

Métodos:
- \+ exibirdica( ): void

Construtor:
- \+ Europa(String nomecategoria)

public class Aves extends Animais

Atributos:
- \+ (super) nomecategoria: String

Métodos:
- \+ exibirdica( ): void

Construtor:
- \+ Aves(String nomecategoria)

public class Mamifero extends Animais

Atributos:

+ (super) nomecategoria: String

Métodos:

+ exibirdica( ): void

Construtor:

+ Mamifero(String nomecategoria)

public class Marinho extends Animais

Atributos:

+ (super) nomecategoria: String

Métodos:

+ exibirdica( ): void

Construtor:

+ Marinho(String nomecategoria)

public class Animacao extends Personagem

Atributos:

+ (super) nomecategoria: String

Métodos:

+ exibirdica( ): void

Construtor:

+ Animacao(String nomecategoria)

public class SuperHeroi extends Personagem

Atributos:

+ (super) nomecategoria: String

Métodos:

+ exibirdica( ): void

Construtor:

+ Personagem(String nomecategoria)

public class Vilao extends Personagem

Atributos:

+ (super) nomecategoria: String

Métodos:

+ exibirdica( ): void

Construtor:

+ Vilao(String nomecategoria)

As classes *America*, *Asia* e *Europa* herdam os atributos e métodos da classe *Paises*, detalhada anteriormente. Implementado na classe mãe, o método *exibirCategoria( )* não requer *override* nas três classes filhas. No entanto, o método *exibirDica( )*, abstrato tanto em *Paises* quanto em *Categoria*, foi implementado de forma a retornar uma dica válida para a categoria em questão. No caso da classe *America*, por exemplo, o método *exibirDica* retorna a *String* “Este país é da América”. No caso da classe *Asia*, o método retorna a *String* “Este país é da Ásia”, e assim por diante.

Da mesma forma, *Aves*, *Mamifero* e *Marinho* herdam atributos e métodos da classe *Animais*, e as classes *Animacao*, *SuperHeroi* e *Vilao* herdam atributos e métodos de *Personagem*. Todas elas podem ser instanciadas, já que não são abstratas, e implementam suas próprias versões de *exibirDica*, para que retornem dicas válidas ao jogador.

Teste

Todos os testes feitos neste trabalho foram desenvolvidos durante a implementação do aplicativo - validando métodos e atributos das classes principais - com o auxílio da ferramenta JUnit, ensinada previamente na cadeira.
Para melhor organização, esta parte será dividida em seções, cada uma se referindo aos testes feitos em determinada classe, dentre elas: classe Jogo, classe Categoria e classe Palavra.

- CategoriaTeste

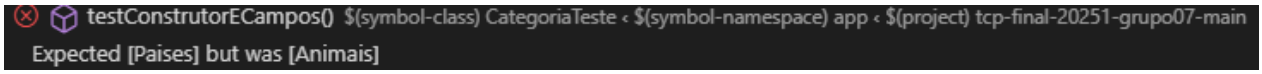
Após a criação da classe Categoria, a fim de testar o seu funcionamento, foi criado um teste que verifica se a chamada de seu construtor corretamente instancia uma nova categoria. Quando chamado com um caso de teste correto, indica que não houve erros. Caso contrário, avisa que algum atributo ou método não é igual ao esperado.

```
@Test
public void testConstrutorECampos() {
    Categoria categoria = new Aves(nomecategoria:"Animais");

    assertEquals("Animais", categoria.nomecategoria);
    assertEquals("Categoria: Animais", categoria.exibirCategoria());
    assertEquals("Este animal é uma ave.", categoria.exibirDica());
}
```


```java
@Test
public void testConstrutorECampos() {
    Categoria categoria = new Aves(nomecategoria:"Animais");

    assertEquals("Paises", categoria.nomecategoria);
    assertEquals("Categoria: Animais", categoria.exibirCategoria());
    assertEquals("Este animal é uma ave.", categoria.exibirDica());
}
```

- PalavraTeste

Após a criação da classe Palavra, a fim de testar o seu funcionamento, foram criados vários testes que verificam o funcionamento de seus métodos e construtores. Quando chamados com casos de testes corretos, indicam que não houve erros. Caso contrário, avisam que algum atributo ou método não é igual ao esperado. Abaixo seguem esses testes, cuja motivação é explicitada em seus títulos, bem como os casos em que devem apresentar êxito ou falhas.

```java
@Test
public void testCriacaoPalavra() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Brasil", numLetras:6, categoria);

    assertEquals("Brasil", palavra.getStringPalavra());
    assertEquals(6, palavra.getNumLetras());
}
```

```java
@Test
public void testCriacaoPalavra() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Brasil", numLetras:6, categoria);

    assertEquals("Brasil", palavra.getStringPalavra());
    assertEquals(2, palavra.getNumLetras());  ⊗ 6 != 2
}
```

⊗ testCriacaoPalavra() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main
Expected [2] but was [6]

```java
@Test
public void testContemLetraPresente() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Argentina", numLetras:9, categoria);

    assertTrue(palavra.contemLetra(letra:'A') || palavra.contemLetra(letra:'a'));
}
```

✓ testContemLetraPresente() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testContemLetraPresente() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Argentina", numLetras:9, categoria);

    assertTrue(palavra.contemLetra(letra:'B') || palavra.contemLetra(letra:'b'));  ⊗ false != true
}
```

⊗ testContemLetraPresente() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main
Expected [true] but was [false]

```java
@Test
public void testContemLetraAusente() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Argentina", numLetras:9, categoria);

    assertFalse(palavra.contemLetra(letra:'z'));
}
```

✓ testContemLetraAusente() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testContemLetraAusente() {
    Categoria categoria = new America(nomecategoria:"Países");
    Palavra palavra = new Palavra(palavra:"Argentina", numLetras:9, categoria);

    assertFalse(palavra.contemLetra(letra:'a')); ⊗ true != false
}
```

testContemLetraAusente() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

Expected [false] but was [true]

```java
@Test
public void testEscolhePalavraSecreta() {
    Categoria categoria = new America(nomecategoria:"Países");
    List<Palavra> lista = new ArrayList<>();
    lista.add(new Palavra(palavra:"Chile", numLetras:5, categoria));
    lista.add(new Palavra(palavra:"Peru", numLetras:4, categoria));

    Palavra secreta = Palavra.escolhePalavraSecreta(lista);

    assertNotNull(secreta);
    assertTrue(lista.contains(secreta));
}
```

testEscolhePalavraSecreta() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testEscolhePalavraSecreta() {
    Categoria categoria = new America(nomecategoria:"Países");
    List<Palavra> lista = new ArrayList<>();
    lista.add(new Palavra(palavra:"Chile", numLetras:5, categoria));
    lista.add(new Palavra(palavra:"Peru", numLetras:4, categoria));

    Palavra secreta = Palavra.escolhePalavraSecreta(lista);
    Palavra naotanalista = new Palavra(palavra:"Brasil", numLetras:6, categoria);

    assertNotNull(secreta);
    assertTrue(lista.contains(naotanalista)); ⊗ false != true
}
```

testEscolhePalavraSecreta() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

Expected [true] but was [false]

```java
@Test
public void testEscolhePalavraSecretaListaVazia() {
    List<Palavra> lista = new ArrayList<>();
    Palavra secreta = Palavra.escolhePalavraSecreta(lista);
    assertNull(secreta);
}
```

testEscolhePalavraSecretaListaVazia() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testEscolhePalavraSecretaListaNull() {
    Palavra secreta = Palavra.escolhePalavraSecreta(lista:null);
    assertNull(secreta);
}
```

testEscolhePalavraSecretaListaNull() $(symbol-class) PalavraTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

- JogoTeste

Após a criação da classe Jogo, a fim de testar o seu funcionamento, foram criados vários testes que verificam o funcionamento de seus métodos e construtores. Quando chamados com casos de testes corretos, indicam que não houve erros. Caso contrário, avisam que algum atributo ou método não é igual ao esperado. Abaixo seguem esses testes, cuja motivação é explicitada em seus títulos, bem como os casos em que devem apresentar êxito ou falhas.

```java
@Test
public void testInicializacaoJogo() {
    assertEquals(6, jogo.getNumvidas(), "Jogo deve começar com 6 vidas");
    assertNotNull(jogo.getPalavra(), "Palavra não deve ser nula");
    assertTrue(jogo.getLetrasTestadas().isEmpty(), "Nenhuma letra deve ter sido testada");
}
```

testInicializacaoJogo() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testInicializacaoJogo() {
    assertEquals(9, jogo.getNumvidas(), "Jogo deve começar com 6 vidas"); ⊗ 6 != 9
    assertNotNull(jogo.getPalavra(), "Palavra não deve ser nula");
    assertTrue(jogo.getLetrasTestadas().isEmpty(), "Nenhuma letra deve ter sido testada");
}
```

testInicializacaoJogo() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testAcertarLetraCorreta() {
    String palavra = jogo.getStringPalavra();
    char letraCorreta = palavra.charAt(index:0); // pega a primeira letra da palavra secreta

    int resultado = jogo.checaLetra(letraCorreta);
    assertEquals(1, resultado, "Deveria retornar 1 para letra correta");
    assertTrue(jogo.getLetrasTestadas().contains(letraCorreta), "Letra correta deveria estar nas testadas");
    assertEquals(6, jogo.getNumvidas(), "Número de vidas não deve mudar em acerto");
}
```

testAcertarLetraCorreta() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

```java
@Test
public void testAcertarLetraCorreta() {
    String palavra = jogo.getStringPalavra();
    char letraCorreta = palavra.charAt(index:0); // pega a primeira letra da palavra secreta

    int resultado = jogo.checaLetra(letraCorreta);
    assertEquals(-1, resultado, "Deveria retornar 1 para letra correta");
    assertTrue(jogo.getLetrasTestadas().contains(letraCorreta), "Letra correta deveria estar nas testadas");
    assertEquals(6, jogo.getNumvidas(), "Número de vidas não deve mudar em acerto");
}
```

testAcertarLetraCorreta() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-grupo07-main

Expected [-1] but was [1]

org.opentest4j.AssertionFailedError: Deveria retornar 1 para letra correta ==] expected: [-1] but was: [1] at app.JogoTeste.

```java
@Test
public void testLetraRepetidaNaoAlteraEstado() {
    char letra = jogo.getStringPalavra().charAt(index:0);
    jogo.checaLetra(letra); // primeira vez
    int resultadoSegundaTentativa = jogo.checaLetra(letra); // repetida

    assertEquals(0, resultadoSegundaTentativa, "Deveria retornar 0 para letra repetida");
    assertEquals(6, jogo.getNumvidas(), "Vidas não devem diminuir ao repetir letra já testada");
}
```

testLetraRepetidaNaoAlteraEstado() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-

```java
@Test
public void testLetraRepetidaNaoAlteraEstado() {
    char letra = jogo.getStringPalavra().charAt(index:0);
    jogo.checaLetra(letra); // primeira vez
    int resultadoSegundaTentativa = jogo.checaLetra(letra); // repetida

    assertEquals(0, resultadoSegundaTentativa, "Deveria retornar 0 para letra repetida");
    assertEquals(5, jogo.getNumvidas(), "Vidas não devem diminuir ao repetir letra já testada"); ⊗ 6 != 5
}
```

testLetraRepetidaNaoAlteraEstado() $(symbol-class) JogoTeste ‹ $(symbol-namespace) app ‹ $(project) tcp-final-20251-gr

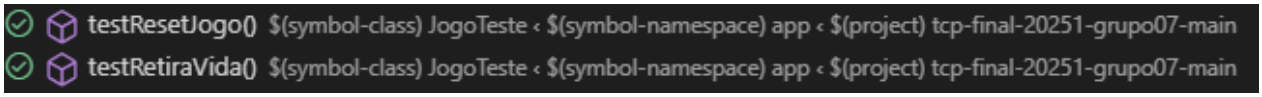
Expected [5] but was [6]

org.opentest4j.AssertionFailedError: Vidas não devem diminuir ao repetir letra já testada ==] expected: [5] but was: [6] a

```java
@Test
public void testRetiraVida() {
    jogo.retiraVida();
    assertEquals(5, jogo.getNumvidas(), "Após retirar vida, deve ter 5 vidas restantes");
}

@Test
public void testResetJogo() {
    jogo.checaLetra(letra:'Z'); // erra de propósito
    jogo.checaLetra(letra:'X');
    assertEquals(4, jogo.getNumvidas(), "Após errar duas letras, vidas devem ser 4");

    jogo.reset();
    assertEquals(6, jogo.getNumvidas(), "Após reset, deve voltar a 6 vidas");
    assertTrue(jogo.getLetrasTestadas().isEmpty(), "Após reset, letras testadas devem estar vazias");
    assertNotNull(jogo.getPalavra(), "Após reset, palavra não deve ser nula");
}
```



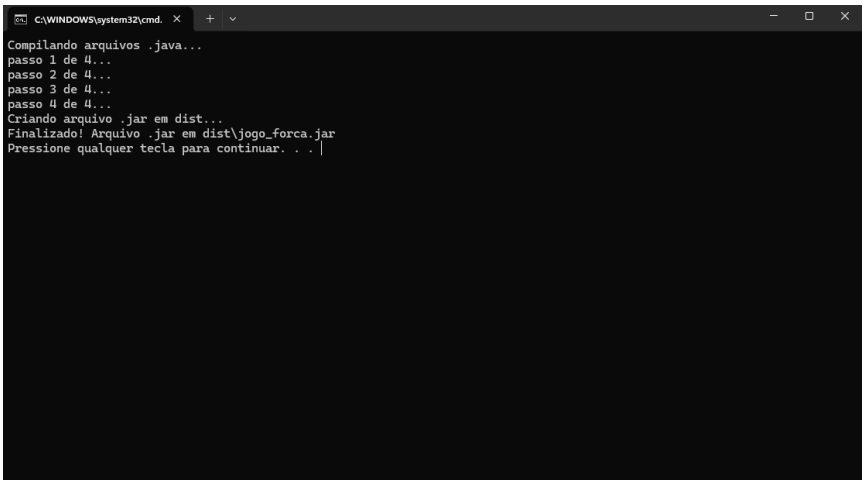

```java
@Test
public void testRetiraVida() {
    jogo.retiraVida();
    assertEquals(2, jogo.getNumvidas(), "Após retirar vida, deve ter 5 vidas restantes"); ⊗ 5 != 2
}
```

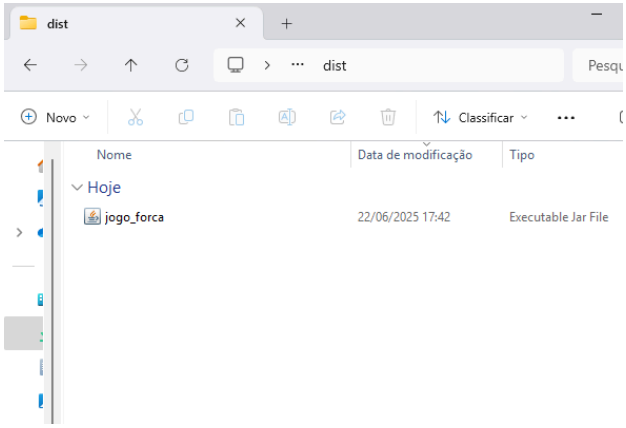
```java
@Test
public void testResetJogo() {
    jogo.checaLetra(letra:'Z'); // erra de propósito
    jogo.checaLetra(letra:'X');
    assertEquals(5, jogo.getNumvidas(), "Após errar duas letras, vidas devem ser 4"); ⊗ 4 != 5

    jogo.reset();
    assertEquals(6, jogo.getNumvidas(), "Após reset, deve voltar a 6 vidas");
    assertTrue(jogo.getLetrasTestadas().isEmpty(), "Após reset, letras testadas devem estar vazias");
    assertNotNull(jogo.getPalavra(), "Após reset, palavra não deve ser nula");
}
```

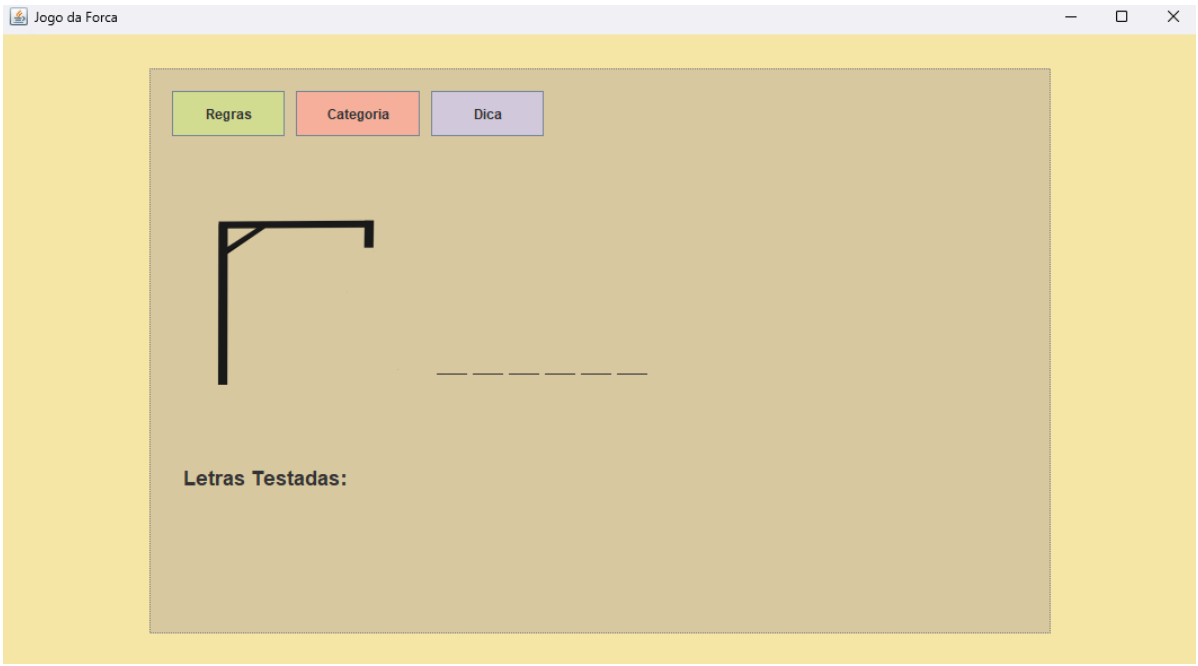
Executável

A execução da aplicação ocorre através da compilação e geração do arquivo *.jar* do jogo. Para a geração do *.jar*, foi elaborado um *batch file* (*compile.bat*) na pasta raiz do projeto, que executa a sequência de comandos responsável pela compilação e geração do executável no prompt de comando. O arquivo executável é gerado no diretório do projeto, na pasta "*dist*".

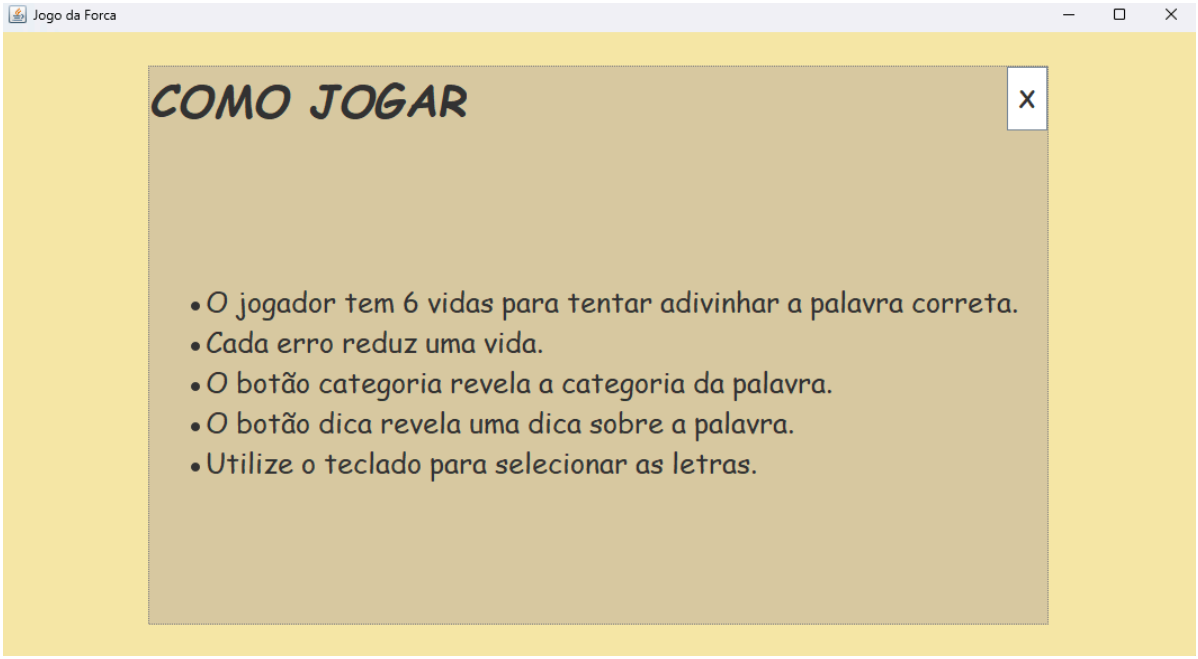


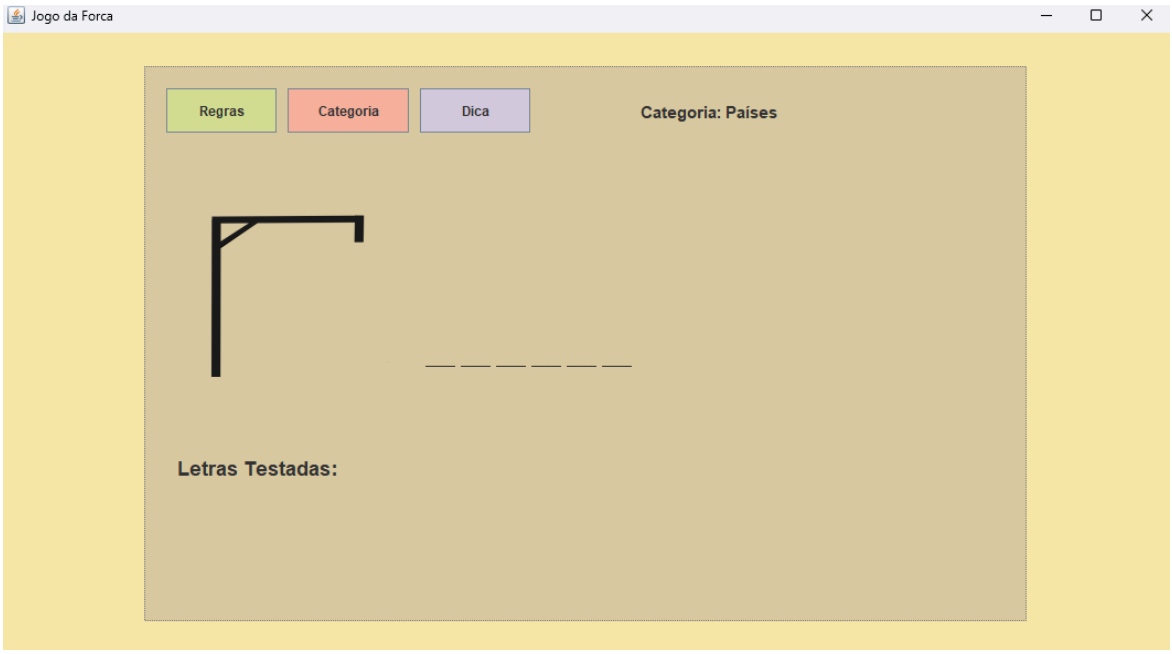


A execução do arquivo *jogo_forca.jar* dá início ao jogo. Uma palavra aleatória do banco de palavras é escolhida pela aplicação, e o usuário já pode realizar o palpite de uma letra usando o teclado.



Alternativamente, o usuário também pode acessar a tela de regras a qualquer momento, clicando no botão “Regras”, verificar a categoria da palavra a ser adivinhada, no botão “Categoria”, ou ainda, pedir uma dica relacionada à palavra, no botão “Dica”.





Sempre que ocorre o palpite de uma letra, ela é adicionada ao banco de letras testadas, na seção inferior da tela de jogo. Caso a letra não esteja presente na palavra, o jogador perde uma vida, fato simbolizado por uma parte do personagem de palito sendo adicionada à forca. Caso a letra esteja correta, ela é adicionada às suas correspondentes lacunas na palavra.

Caso o jogador perca seis vidas, ocorre o fim de jogo, sinalizado pela tela de derrota. A tela de derrota revela a palavra correta, com a mensagem “Você Perdeu”. Caso o jogador adivinhe todas as letras da palavra secreta antes que suas seis vidas cheguem ao fim, o fim de jogo é sinalizado pela tela de vitória, que mostra a palavra adivinhada com a mensagem “Você Ganhou”. Após o fim de jogo, o jogador pode dar início a um novo jogo, clicando em “Reiniciar”, ou fechar a aplicação, clicando em “Sair”.


Jogo da Forca
VOCÊ GANHOU!
A palavra era: MÉXICO
REINICIAR
SAIR