**UNIVERSIDADE FEDERAL DE SERGIPE**

**SISTEMAS DA INFORMAÇÃO**

KAUAN FELIPE

PAULO HENRIQUE

LEANDRO MENDONÇA

MATHEUS ANDRADE

IGOR AUGUSTO

**DOCUMENTACAO PROJETO HASKELL - LP**

Jogo Termo

**Itabaiana - SE**

**2025**

**UNIVERSIDADE FEDERAL DE SERGIPE**  
**SISTEMAS DE INFORMAÇÃO**

**KAUAN FELIPE**  
**PAULO HENRIQUE**  
**LEANDRO MENDONÇA**  
**MATHEUS ANDRADE**  
**IGOR AUGUSTO**

**DOCUMENTAÇÃO PROJETO HASKELL - LP**  
**JOGO TERMO**

Projeto apresentado ao Curso de Sistemas de Informação da Universidade Federal de Sergipe como requisito parcial para obtenção da nota da 1ª unidade.

Professor : Antônio Alves

**Itabaiana - SE**  
**2025**

**SUMÁRIO**

1. **Introdução**
2. **Como Funciona o Jogo**
3. **BancoDePalavras.txt**
4. **Grafico.hs  
    4.1. Descrição do Módulo  
    4.2. AplicarCor  
    4.3. AplicarCorLetra  
    4.4. ImprimirLista**
5. **Jogo.hs  
    5.1. Função Jogo**
6. **Logica.hs  
    6.1. RecolherPalavras  
    6.2. TamanhoPalavra  
    6.3. TransformarMinusculo  
    6.4. TaNaLista  
    6.5. VerificarResultado  
    6.6. VerificarPalavra  
    6.7. CriaLista  
    6.8. CriarTentativasVazias  
    6.9. AtualizarTentativas  
    6.10. EscolherPalavraPeloIndice  
    6.11. GerarAleatoria  
    6.12. ContaPalavras**
7. **Tipos.hs**
8. **Main.hs**

**·**

1. **Introdução**

Esse documento visa mostrar a estrutura, métodos e lógicas aplicadas na implementação do jogo Termo. Utilizando conceitos aplicados na aula de Linguagem de Programação do curso de Sistemas da Informação. Este trabalho foi feito e implementado com a linguagem de programação Haskell, separados em 5 módulos e 1 arquivo de texto, são eles:

**-BancoDePalavras.txt**

**-Grafico.hs**

**-Jogo.hs**

**-Logica.hs**

**-Tipos.hs**

**-Main.hs**

1. **Como funciona o jogo**

O jogo Termo é um jogo de palavras de adivinhação cujo objetivo é descobrir uma palavra secreta de 5 letras dentro de 6 tentativas. A cada tentativa, o jogador deve digitar uma palavra de 5 letras e o jogo dá um feedback visual usando cores para indicar quais letras estão corretas, incorretas ou parcialmente corretas. Se uma letra estiver correta e na posição certa, ela é destacada em verde. Se a letra estiver na palavra, mas na posição errada, ela aparece em amarelo. Já se a letra não faz parte da palavra, ela é marcada em cinza. O jogador deve usar essas pistas para ajustar suas próximas tentativas e tentar adivinhar a palavra secreta dentro do número limitado de tentativas.

1. **BancoDePalavras.txt**

**Descrição**: Arquivo de texto contendo todas as palavras de 5 letras do dicionário em português. Todas as palavras estão com letras minúsculas e com caracteres especiais como acentos e cedilhas.

1. **Grafico.hs**

**Descrição:** Módulo criado para desenvolver a parte gráfica e visual do projeto. Nela é possível aplicar as cores aos caracteres que forem lidos como certo, errado ou parcial, além de outras funções como limpar a tela e imprimir a palavra-tentativa que o usuário digitou.

**Conteúdo Utilizado:** Funções, Função de alta ordem, Listas, Função Map, List Comprehension.

* 1. **AplicarCor**

A função aplicarCor recebe dois parâmetros: um valor do tipo Resultado (que pode ser Correta, Errado ou Parcial) e um caractere. Dependendo do valor de Resultado, ela retorna a letra com uma cor diferente. Se o resultado for Correta, a letra é colorida de verde, se for Errado, a letra é mantida sem cor, e se for Parcial, a letra é colorida de amarelo.

* 1. **AplicarCorLetra**

Função que aplica a função aplicarCor a uma tupla que contém um Resultado e uma letra, retornando a letra com a cor correspondente.

* 1. **ImprimirLista**

Recebe uma lista de palavras e utiliza a função mapM\_ para imprimir cada palavra da lista, chamando putStrLn para exibir as palavras.

1. **Jogo.hs**

**Descrição:** Modulo onde vai rodar o jogo, detém toda a lógica de loops e tentativas para o jogo continuar rodando.

**Conteudo Utilizado:** Funções, Listas.

* 1. **Jogo**

Função que recebe a palavra-tentativa do input do usuário e gera um loop de acordo com o número de tentativas do jogador, com um total de 6 tentativas. A função se baseia no número de tentativas e no acerto do jogador, se o número de tentativas chegar a zero o jogo acaba e mostra na tela q o jogador perdeu. Além disso, a cada tentativa do jogador irá mostrar na tela seus erros e acertos em cada letra colorida. Se o jogador tentar digitar uma palavra com mais de 5 letras é notificado ao jogador e ele não perde a sua tentativa. Quando o jogador acertar a palavra ou esgote o número de tentativas, o jogo acaba.

1. **Logica.hs**

**Descrição:** Módulo onde apresenta toda a lógica do jogo, com funções para tratamento de palavras, verificações, list comprehension, geração de escolhas aleatórias, etc.

**Conteúdo Utilizado:** Lista, List Comprehension, Map, Polimorfismo, Recursividade.

* 1. **RecolherPalavras**

A função recebe um caminho de arquivo como argumento e retorna o conteúdo desse arquivo em formato de string.

* 1. **TamanhoPalavra**

Calcula o tamanho de uma lista, utilizando List Comprehension para contar cada elemento até chegar ao final da lista, retornando o número de elementos.

* 1. **TransformarMinusculo**

Recebe uma string e utiliza a função map e a toLower para transformar todas as letras da string em minúsculas.

* 1. **TaNaLista**

A função taNaLista verifica se um determinado elemento está presente em uma lista. Ela faz isso de forma recursiva, comparando o elemento atual da lista com o valor procurado. Se encontrar, retorna True, e se não, continua a busca na próxima posição da lista, retornando False caso o elemento não seja encontrado.

* 1. **VerificarResultado**

A função compara duas palavras, uma sendo a palavra correta e a outra a tentativa. Ela utiliza a função aplicarCorLetra para gerar uma string com as cores apropriadas para cada letra, dependendo do resultado da comparação. Se as letras forem iguais, a função marca a letra como Correta. Se a letra da tentativa estiver presente na palavra correta, mas em uma posição diferente, ela é marcada como Parcial. Caso contrário, a letra é considerada Errada.

* 1. **VerificarPalavra**

A função verificarPalavra compara duas palavras, gerando uma lista de resultados (Corretas, Parcial ou Errado) para cada letra utilizando zipWith.

* 1. **CriaLista**

A função criaLista cria uma lista de palavras a partir de uma string, separando as palavras por um caractere específico. Quando encontra esse caractere, ela começa a formar uma nova palavra, ignorando os espaços em branco.

* 1. **CriarTentativasVazias**

Cria uma lista de tentativas vazias, representadas como uma string com 5 underlines e espaços ("\_ \_ \_ \_ \_") repetida um número de vezes especificado pelo parâmetro de entrada.

**6.9. AtualizarTentativas**

A função atualizarTentativas atualiza uma lista de tentativas, substituindo uma tentativa vazia pela comparação entre a palavra correta e a tentativa feita.

**6.10. EscolherPalavraPeloIndice**

Essa função seleciona uma palavra de uma lista com base no índice fornecido. Se o índice for encontrado, a função retorna a palavra correspondente.

**6.11. GerarAleatorio**

A função gera um número aleatório dentro de um intervalo especificado, baseado no tempo atual. Ela utiliza a função getCurrentTime para pegar o tempo em segundos e usa esse valor para gerar o número aleatório.

**6.12. ContaPalavra**

Essa função percorre uma lista de listas de caracteres e conta quantas palavras existem nela, incrementando a contagem a cada palavra encontrada.

1. **Tipos.hs**

**Descrição:** Esse modulo serviu para armazenar os types e os datas do projeto. Foi aplicado o tipo Palavra para ser usado como a palavra a ser descoberta pelo jogador, o tipo Jogo que é uma tupla com a Palavra e uma lista dessa Palavra, além de ter um data Resultado que pode ter 3 valores Correta, Errada ou Parcial.

**Conteúdo Utilizado:** Tipos algébricos, Tuplas, Listas.

1. **Main.hs**

**Descrição:** Nesse modulo é onde o jogo vai iniciar, faremos a leitura das palavras, divisão das palavras separadas por vírgulas no arquivo de texto, gera um índice aleatório e sorteia a palavra da rodada.

**Conteúdo Utilizado:** Funções, Arquivo.