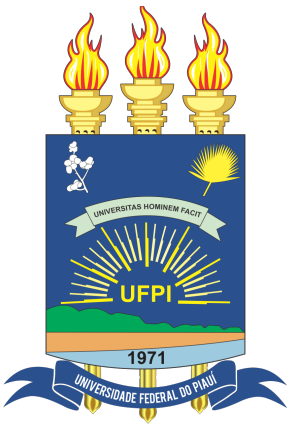

Universidade Federal Do Piauí

Sistemas De Informação

Programação Funcional

4º - 7º Período

Relatório (Trabalho Final)

Alunos: Anderson Luz, Hermeson Alves, Marcos William, Raildom da Rocha, João Marcos Rufino
Professor: Júlio Vítor Monteiro Marques

Junho de 2025

Contents

1 Introdução

Este relatório apresenta uma visão geral do jogo da Forca desenvolvido em *Haskell*, abordando as principais decisões de design, a estrutura do projeto, o processo de compilação e exemplos práticos de uso. O sistema está organizado em torno dos arquivos `Main.hs`, `Tipos.hs`, `Ranking.hs`, `Utilitarios.hs` e `LogicaJogo.hs`, com o objetivo de esclarecer as escolhas que orientaram a arquitetura do jogo, descrever a organização do código, explicar como executá-lo e ilustrar, por meio de exemplos, a experiência do jogador.
A motivação por trás deste trabalho surgiu do anseio por desenvolver algo que proporcionasse entretenimento e diversão durante sua criação e uso. Em vez de seguir o caminho tradicional de projetos acadêmicos repetitivos, optamos por construir uma experiência interativa que fugisse da monotonia característica de muitos trabalhos universitários, geralmente mecânicos e sem graça. Com isso, buscamos tornar o processo mais envolvente e criativo.

2 Escolhas de Design

O projeto foi estruturado com foco em modularidade, encapsulamento e programação funcional, aproveitando as características do Haskell para criar um código robusto, reutilizável e fácil de manter. A seguir, são apresentadas as principais escolhas de design:

2.1 Modularidade e Separação de Responsabilidades

- O projeto está dividido em cinco módulos, cada um com uma responsabilidade específica:
  - `Main.hs`: Coordena o fluxo do jogo e a interação com o usuário (I/O).
  - `Tipos.hs`: Define os tipos de dados fundamentais, como o tipo abstrato `Jogo`.
  - `Ranking.hs`: Gerencia a persistência de dados (histórico e ranking).
  - `Utilitarios.hs`: Fornece funções auxiliares, como seleção de palavras.
  - `LogicaJogo.hs`: Contém a lógica pura do jogo, separada do I/O.
- Essa separação segue o princípio de responsabilidade única, facilitando manutenção e testes.

2.2 Encapsulamento com Tipo Abstrato de Dados

- O tipo `Jogo` em `Tipos.hs` é um tipo abstrato de dados, com o construtor `JogoInternal` não exportado, forçando o uso de funções de interface (`criarJogoInicial`, `palavraSecretaJogo`, etc.).
- Isso protege a integridade do estado do jogo e facilita alterações futuras.

2.3 Programação Funcional

- **Imutabilidade**: O estado do jogo é atualizado criando novas instâncias de `Jogo`.
- **Funções Puras**: A lógica em `LogicaJogo.hs` é puramente funcional, separada do I/O.
- **Monads**: Operações de I/O são gerenciadas com `IO` de forma disciplinada.
- **Recursão**: Funções como `pedirNomeJogador` usam recursão para entradas inválidas.

2.4 Tratamento de Erros

- Operações de I/O são tratadas com `catchIOError` (`Utilitarios.hs`, `Ranking.hs`).
- `encontrarArquivo` tenta múltiplos caminhos para o arquivo de palavras, usando palavras de emergência em caso de falha.
- Parsing robusto com `readMaybe` em `Ranking.hs` lida com arquivos corrompidos.

2.5 Persistência de Dados

- Arquivos de texto (`historico_partidas.txt`, `ranking_acumulado.txt`) armazenam dados, adequados para a simplicidade do jogo.
- Parsing robusto garante resiliência a erros nos arquivos.

2.6 Interface do Usuário

- Interface de terminal com códigos ANSI (`limparTela`) para melhorar a experiência visual.
- A classe `Exibivel` formata a exibição do jogo (boneco da forca, palavra parcial, etc.).

3 Estrutura do Projeto

O projeto segue uma organização modular, com cada módulo desempenhando um papel específico e interagindo de forma controlada com os demais. Abaixo está uma descrição detalhada da estrutura, incluindo os papéis, funções principais, dependências e interações de cada módulo, bem como os arquivos de dados utilizados.

3.1 Módulos do Projeto

- `Main.hs`:
  - **Papel**: Orquestra o fluxo principal do jogo, gerenciando a interação com o usuário via terminal e coordenando chamadas aos outros módulos.
  - **Funções Principais**:
    - `main`: Ponto de entrada do programa, inicializa o diretório `data` e exibe a mensagem de boas-vindas.
    - `pedirNomeJogador`: Solicita e valida o nome do jogador.
    - `iniciarNovoJogo`: Seleciona uma palavra aleatória e inicia um novo jogo.
    - `loopJogo`: Gerencia o loop principal do jogo, exibindo o estado e processando entradas.
    - `processarEntrada`: Analisa entradas do usuário (letra ou palavra completa).
    - `tentarPalavra`: Processa tentativas de adivinhar a palavra inteira.
    - `finalizarJogo`: Exibe o resultado final e salva a pontuação.
    - `jogarNovamente`: Pergunta se o jogador deseja continuar.
  - **Dependências**: Depende de `Tipos.hs` (para `Jogo`, `EstadoJogo`, `Exibivel`), `LogicaJogo.hs` (para `chutarLetra`, `calcularPontuacao`), `Utilitarios.hs` (para `selecionarPalavra`, `limparTela`) e `Ranking.hs` (para `salvarPontuacao`, `exibirRankingGeral`, `exibirHistoricoPartidas`).
  - **Interações**: Chama `selecionarPalavra` para obter uma palavra, usa `criarJogoInicial` para inicializar o estado, atualiza o jogo com `chutarLetra` ou `tentarPalavra`, e persiste resultados com `salvarPontuacao`.

- `Tipos.hs`:
  - **Papel**: Define os tipos de dados fundamentais e a interface para manipulação do estado do jogo, implementando encapsulamento via tipo abstrato.
  - **Funções/Tipos Principais**:
    * `Jogo`: Tipo abstrato que encapsula o estado do jogo (palavra secreta, letras chutadas, tentativas restantes, estado do jogo).
    * `EstadoJogo`: Enumeração com os estados `Jogando`, `Ganhou` e `Perdeu`.
    * `Exibivel`: Classe para formatação de exibição, com instância para `Jogo`.
    * `criarJogoInicial`, `palavraSecretaJogo`, `letrasChutadasJogo`, `tentativasRestantesJogo`, `estadoJogoJogo`: Funções de interface para criar e acessar o estado do jogo.
    * `construirJogoComNovosValores`: Função fábrica para criar novos estados.
    * `desenharBoneco`: Gera representações ASCII do boneco da forca.
  - **Dependências**: Usa bibliotecas padrão (`Data.List`, `Data.Char`) para manipulação de strings e listas.
  - **Interações**: É usado por todos os outros módulos como base para manipulação do estado do jogo. Fornece a estrutura de dados central e métodos de exibição.
- `Ranking.hs`:
  - **Papel**: Gerencia a persistência de dados (histórico de partidas e ranking acumulado) e exibe informações de pontuação.
  - **Funções Principais**:
    * `salvarPontuacao`: Salva o resultado de uma partida no histórico e atualiza o ranking.
    * `lerHistoricoPartidas`, `lerRankingAcumulado`: Lêem dados dos arquivos de texto.
    * `exibirRankingGeral`, `exibirHistoricoPartidas`: Exibem o ranking e o histórico.
    * `parseLinhaPontuacao`, `parseLinhaHistorico`: Fazem parsing dos arquivos.
    * `atualizarPontuacaoJogador`: Atualiza a pontuação acumulada de um jogador.
  - **Dependências**: Depende de `Tipos.hs` (para `Jogo`, `EstadoJogo`), `LogicaJogo.hs` (para `calcularPontuacao`) e `Utilitarios.hs` (para `trim`).
  - **Interações**: Interage com `Main.hs` para salvar e exibir pontuações, usa `calcularPontuacao` para determinar pontos e manipula arquivos de texto para persistência.
- `Utilitarios.hs`:
  - **Papel**: Fornece funções auxiliares para manipulação de arquivos, seleção de palavras aleatórias e limpeza do terminal.
  - **Funções Principais**:
    * `selecionarPalavra`: Seleciona uma palavra aleatória do banco de palavras.
    * `lerBancoPalavras`, `lerPalavrasDoArquivo`: Lêem palavras de `palavras.txt`.
    * `inicializarArquivoPalavras`, `criarArquivoPalavras`: Gerenciam a criação do arquivo de palavras.
    * `limparTela`: Limpa o terminal com códigos ANSI.
    * `trim`: Remove espaços em branco de strings.

  - **Dependências**: Usa bibliotecas padrão (`System.IO`, `System.Random`, `Data.Char`) para I/O e manipulação de strings.
  - **Interações**: Fornece `selecionarPalavra` para `Main.hs` iniciar o jogo, `limparTela` para melhorar a interface e `trim` para `Ranking.hs` processar strings.

- `LogicaJogo.hs`:
  - **Papel**: Contém a lógica pura do jogo, isolada de operações de I/O, para facilitar testes e manutenção.
  - **Funções Principais**:
    * `chutarLetra`: Processa o chute de uma letra, atualizando o estado do jogo.
    * `atualizarEstadoJogo`: Determina o estado do jogo após um chute.
    * `letraValida`: Valida se um caractere é uma letra A-Z.
    * `removerEspacos`: Remove espaços de strings para comparação.
    * `calcularPontuacao`: Calcula a pontuação final com base em tentativas restantes e letras corretas.
  - **Dependências**: Depende de `Tipos.hs` (para `Jogo`, `EstadoJogo`, funções de interface) e bibliotecas padrão (`Data.Char`, `Data.List`).
  - **Interações**: Fornece lógica para `Main.hs` processar chutes e para `Ranking.hs` calcular pontuações.

3.2 Arquivos de Dados

- `data/palavras.txt`: Armazena a lista de palavras disponíveis para o jogo, lida por `Utilitarios.hs`. Cada linha contém uma palavra, convertida para maiúsculas.
- `data/historico_partidas.txt`: Registra detalhes de cada partida (nome, pontuação, resultado, palavra secreta), manipulado por `Ranking.hs`.
- `data/ranking_acumulado.txt`: Armazena a pontuação total acumulada por jogador, também gerenciado por `Ranking.hs`.

3.3 Dependências e Fluxo de Dados

- `Main.hs` é o ponto central, dependendo de todos os módulos para coordenar o fluxo do jogo.
- `Tipos.hs` é a base estrutural, fornecendo tipos e interfaces usados por todos os outros módulos.
- `LogicaJogo.hs` depende de `Tipos.hs` para manipular o estado do jogo e fornece lógica para `Main.hs` e `Ranking.hs`.
- `Ranking.hs` depende de `Tipos.hs` (para `Jogo`), `LogicaJogo.hs` (para `calcularPontuacao`) e `Utilitarios.hs` (para `trim`).
- `Utilitarios.hs` é independente, mas suas funções são usadas por `Main.hs` e `Ranking.hs`.
- **Fluxo de Dados**:
  - `Utilitarios.hs` fornece palavras para `Main.hs` via `selecionarPalavra`.
  - `Main.hs` inicializa o jogo com `Tipos.hs` (`criarJogoInicial`) e atualiza o estado com `LogicaJogo.hs` (`chutarLetra`).

  - `Ranking.hs` persiste resultados de `Main.hs` e usa `LogicaJogo.hs` para calcular pontuações.
  - `Tipos.hs` fornece a exibição formatada (`Exibivel`) para `Main.hs`.

3.4 Organização Física

- O projeto assume uma estrutura de diretórios com uma pasta `data/` para armazenar `palavras.txt`, `historico_partidas.txt` e `ranking_acumulado.txt`.
- Os arquivos de código (`.hs`) residem no diretório raiz do projeto, com `Main.hs` servindo como ponto de entrada.
- A modularidade permite que cada arquivo seja compilado independentemente, desde que as dependências sejam resolvidas.

4 Compilação do Código

Para executar o Jogo da Forca, o código Haskell deve ser compilado utilizando o compilador GHC (Glasgow Haskell Compiler). Abaixo, são descritos os métodos de compilação, as dependências externas necessárias e o uso do script `a.bat` para automação.

4.1 Requisitos e Dependências Externas

- **GHC**: O compilador Haskell (versão 8.10 ou superior recomendada) deve estar instalado. Pode ser obtido via Haskell Platform ou GHCup.
- **Bibliotecas Externas**: O projeto utiliza as seguintes bibliotecas, disponíveis no pacote `base` ou como dependências externas:
  - `base`: Inclui `Data.Char`, `Data.List`, `System.IO` e `Control.Exception` (usados em `Tipos.hs`, `Utilitarios.hs`, `Ranking.hs` e `LogicaJogo.hs`).
  - `random`: Usada em `Utilitarios.hs` para seleção de palavras aleatórias (`selecionarPalavra`). Essa biblioteca não está no `base` e deve ser instalada separadamente.
- **Instalação da Biblioteca `random`**:
  - Usando `cabal` (gerenciador de pacotes do Haskell), execute:

```
cabal update
cabal install --lib random
```

  - Alternativamente, com `stack`, adicione `random` ao arquivo `package.yaml` ou execute:

```
stack install random
```

- **Ambiente**: O projeto assume que o diretório `data/` existe no mesmo nível dos arquivos `.hs`, contendo `palavras.txt`. O GHC deve ter acesso de escrita para criar `historico_partidas.txt` e `ranking_acumulado.txt`.

4.2 Compilação Convencional com GHC

O método convencional envolve compilar os arquivos `.hs` usando o GHC diretamente na linha de comando. Siga os passos abaixo:

1. Abra um terminal no diretório raiz do projeto (onde estão `Main.hs`, `Tipos.hs`, etc.).
2. Compile o projeto com o comando:

```
ghc -o jogo_da_forca Main.hs
```

3. Este comando compila `Main.hs` e todos os módulos dependentes (`Tipos.hs`, `Ranking.hs`, `Utilitarios.hs`, `LogicaJogo.hs`), gerando um executável chamado `jogo_da_forca` (ou `jogo_da_forca.exe` no Windows).
4. Execute o jogo com:

```
./jogo_da_forca
```

   (No Windows, use `jogo_da_forca.exe`.)

5. **Notas**:
   - Certifique-se de que o GHC está no `PATH` do sistema.
   - Se a biblioteca `random` não estiver instalada, o GHC exibirá um erro de dependência. Instale-a conforme descrito acima.
   - O comando `ghc` resolve automaticamente as dependências entre os módulos, desde que todos os arquivos `.hs` estejam no mesmo diretório.

4.3 Compilação com o Script `a.bat`

O projeto inclui um arquivo `a.bat` (destinado a sistemas Windows) que automatiza o processo de compilação. Para usar:

1. Certifique-se de que o GHC está instalado e configurado no `PATH`.
2. No diretório raiz do projeto, onde está `a.bat`, abra um terminal (Prompt de Comando ou PowerShell).
3. Execute o script com:

```
a.bat
```

4. O script `a.bat` executa um comando similar a `ghc -o jogo_da_forca Main.hs`, compilando todos os módulos e gerando o executável `jogo_da_forca.exe` e já o executando.
5. **Notas**:
   - O `a.bat` é específico para Windows. Em sistemas Unix/Linux/Mac, um script shell equivalente (e.g., `a.sh`) seria necessário.
   - Se `random` não estiver instalada, a compilação falhará com uma mensagem de erro. Instale a biblioteca conforme descrito.
   - O `a.bat` simplifica o processo, mas assume que os arquivos `.hs` estão no mesmo diretório e que o GHC está configurado corretamente.

4.4 Considerações Adicionais

- **Erros Comuns**:
  - Se o GHC não encontrar `random`, verifique se a biblioteca está instalada globalmente ou no ambiente do projeto.
  - Se o diretório `data/` ou `palavras.txt` estiver ausente, o jogo pode falhar ao iniciar. Crie a pasta `data/` e um arquivo `palavras.txt` com palavras válidas.
- **Execução Alternativa**: Além de compilar com `ghc`, o jogo pode ser executado diretamente com `runhaskell Main.hs`, desde que todas as dependências estejam instaladas. No entanto, isso é mais lento e menos comum para projetos completos.

5 Exemplos de Uso

5.1 Exemplo 1: Iniciando o Jogo e Adivinhando uma Letra

1. O jogador executa o jogo e vê:

```
Bem-vindo ao Jogo da Forca em Haskell!
=====================================
Digite seu nome: Alice
```

2. Após inserir "Alice", o jogo seleciona "HASKELL" e exibe:

```
  ____
  |  |
  |
  |
  |
__|__

Palavra: _ _ _ _ _ _ _
Letras ja utilizadas:
Tentativas restantes: 6
Digite uma letra (A-Z) ou tente adivinhar a palavra inteira:
```

3. O jogador chuta "A", e o estado é atualizado (`chutarLetra`):

```
  ____
  |  |
  |
  |
  |
__|__

Palavra: _ A _ _ _ _ _
Letras ja utilizadas: A
Tentativas restantes: 6
Digite uma letra (A-Z) ou tente adivinhar a palavra inteira:
```

5.2 Exemplo 2: Tentativa de Palavra Completa

1. O jogador tenta "HASKELL" (`tentarPalavra`):
2. O estado muda para `Ganhou`, com pontuação apropriada (`calcularPontuacao`):

```
--- Fim de Jogo ---
    ____
    |  |
    |
    |
    |
  __|__

Palavra: H A S K E L
Letras ja utilizadas: H A S K E L
Tentativas restantes: 6
Parabens, Alice! Voce venceu!
Sua pontuacao final: 90
Partida atual: Alice venceu com 90 pontos
--- Ranking Geral de Jogadores ---
1. Alice - 90 pontos
-----------------------------------
--- Historico de Partidas Recentes ---
1. Alice venceu com 90 pontos
-----------------------------------
Deseja Jogar novamente? (S/N):
```

5.3 Exemplo 3: Perda do Jogo

1. O jogador chuta letras incorretas (“X”, “Y”, “Z”, etc.) até esgotar as 6 tentativas.
2. Após o 6º erro, o estado muda para `Perdeu`: `listado`:

```


--- Fim de Jogo ---
    ____
    |  |
    |  O
    | /|\
    | / \
  __|__

Palavra: _ _ _ _ _ _ _
Letras ja utilizadas: X Y Z Q W V
Tentativas restantes: 0
Que pena, Alice! Voce perdeu.
A palavra secreta era: H A S K E L L
Partida atual jogo: Alice perdeu com 0 pontos
--- Ranking Geral de Jogadores ---
1. Alice - 65 pontos
-----------------------------------
--- Historico de Partidas Recentes ---
1. Alice perdeu com 0 pontos
2. Alice venceu com 65 pontos
-----------------------------------
Deseja jogar novamente? (S/N):
```

5.4 Exemplo 4: Erro de Entrada

1. O jogador digita “@” (`processarEntrada`):

```
Letra invalida. Use apenas uma letra de A-Z.
Pressione Enter para continuar...
```

2. O jogo retorna ao loop principal.