# Relatório Técnico: Implementação do Jogo dos Palitos em Haskell

Artur Welerson Sott Meyer - 202065552C

January 17, 2025

**Abstract**

Este relatório apresenta a implementação de uma solução para o Jogo dos Palitos utilizando a linguagem Haskell. Descrevemos em detalhes a modelagem dos dados, as funções utilizadas e a lógica que implementa estratégias vencedoras, com destaque para a jogada estratégica baseada em propriedades matemáticas do jogo.

## 1 Introdução

O Jogo dos Palitos é um problema clássico que envolve estratégia e raciocínio lógico. Neste jogo, duas partes alternam turnos para remover palitos de fileiras, respeitando regras específicas. O objetivo é forçar o oponente a ser o último a jogar. Este trabalho apresenta uma implementação algorítmica deste jogo em Haskell, contemplando quatro modos de jogo: *fácil*, *difícil*, *difícil com ajuda*, e *máquina contra máquina*.

## 2 Como Rodar o Código

Este projeto foi desenvolvido em Haskell e utiliza o Stack para gerenciamento de dependências e construção do projeto. Aqui estão as etapas para configurar, compilar e rodar o código.

### 2.1 Pré-requisitos

Antes de começar, é necessário ter o `Stack` instalado no seu sistema. `Stack` é uma ferramenta de gerenciamento de projetos Haskell que facilita a construção e execução de programas. Se você ainda não tem o `Stack`, siga as instruções de instalação na documentação oficial.

### 2.2 Clonando o repositório

Primeiro, clone o repositório contendo o código do projeto. Se você ainda não fez isso, pode usar o seguinte comando:

```
git clone https://github.com/ArturMeyer/jogo-dos-palitos-haskell.git
cd <DIRETORIO_DO_PROJETO>
```

## 2.3 Compilando o projeto

Para compilar o projeto, basta executar o comando `stack build` no terminal. Esse comando vai baixar as dependências necessárias e compilar o código.

```
stack build
```

Se o processo de construção for bem-sucedido, você verá uma mensagem indicando que o projeto foi compilado corretamente.

## 2.4 Executando o código

Após a construção do projeto, você pode executar o programa com o comando:

```
stack run
```

Esse comando irá rodar a função principal (`main`) definida no código e iniciar o jogo, onde você poderá interagir com ele diretamente pelo terminal.

## 2.5 Interação com o código no GHCi (opcional)

Se preferir interagir diretamente com o código em um ambiente de REPL (Read-Eval-Print Loop), você pode usar o comando `stack ghci` para carregar o projeto no GHCi.

```
stack ghci
```

Dentro do GHCi, você pode testar funções e expressões diretamente ou carregar módulos específicos do seu projeto.

## 2.6 Finalizando

Após rodar o código, você pode interagir com o jogo e escolher entre diferentes modos de dificuldade e jogabilidade. Caso queira encerrar o programa, basta seguir as instruções no próprio jogo para sair.

# 3 Modelagem dos Dados

A modelagem dos dados é fundamental para estruturar o estado do jogo e as opções disponíveis para o jogador. O modelo de dados principal é composto por dois tipos: `Escolha` e `GameState`.

## 3.1 Tipo `Escolha`

O tipo `Escolha` representa as diferentes opções de dificuldade ou modo de jogo disponíveis para o usuário. Ele é definido como uma soma de tipos, onde cada construtor corresponde a uma opção:

- `Facil`: Representa o modo de jogo fácil.
- `Dificil`: Representa o modo de jogo difícil.
- `MvsM`: Representa o modo em que duas máquinas jogam contra si mesmas (sem interação do usuário).

- `Dificil_Ajuda`: Representa o modo difícil com uma variação de ajuda para o jogador.
- `Sair`: Representa a opção de sair do jogo.

A definição de `Escolha` é a seguinte:

```
data Escolha = Facil | Dificil | Dificil_Ajuda | MvsM | Sair deriving (Show, Read, Eq)
```

Os derivados `Show`, `Read` e `Eq` permitem que o tipo `Escolha` seja convertido para uma representação em string (`Show`), lido a partir de uma string (`Read`) e comparado com outros valores (`Eq`).

### 3.2 Tipo `GameState`

O tipo `GameState` representa o estado do jogo em um dado momento. Ele é composto por dois elementos:

- Uma lista de inteiros (`[Int]`) que representa as fileiras de palitos no jogo.
- Um valor do tipo `Escolha`, que indica a dificuldade ou modo de jogo escolhido pelo jogador.

A definição de `GameState` é a seguinte:

```
type GameState = ([Int], Escolha)
```

A lista de inteiros contém as quantidades de palitos em cada fileira, e o valor do tipo `Escolha` define o modo ou a dificuldade do jogo. O `GameState` é usado para armazenar e atualizar o progresso do jogo, refletindo tanto o estado atual das fileiras de palitos quanto a escolha feita pelo jogador.

## 4 Estratégia Vencedora no Jogo de Palitos

O jogo de palitos é uma instância do conhecido jogo de Nim, onde dois jogadores alternam removendo palitos de várias fileiras. A estratégia vencedora do jogo de Nim é baseada em um conceito matemático chamado *Nim-Sum*, que é a soma binária (XOR) das quantidades de palitos em cada fileira.

### 4.1 Nim-Sum e a Estratégia Vencedora

No jogo de Nim, a estratégia vencedora depende do valor do *Nim-Sum*, que é calculado como o XOR de todos os valores que representam as quantidades de palitos em cada fileira. Se o *Nim-Sum* for diferente de zero, o jogador tem uma jogada vencedora. Caso contrário, se o *Nim-Sum* for igual a zero, o jogador está em uma posição perdedora, assumindo que o adversário jogue perfeitamente.

A máquina difícil, ao calcular o *Nim-Sum*, usa a seguinte lógica:

- Se o *Nim-Sum* não for igual a zero, a máquina calcula a jogada vencedora, que consiste em escolher uma fileira de palitos tal que, ao realizar a jogada, o *Nim-Sum* da configuração do jogo após a jogada se torne zero. Isso é feito através da fórmula:

$$\text{nova quantidade de palitos} = \text{quantidade original} \oplus \text{Nim-Sum}$$

  Onde $\oplus$ representa a operação XOR. A máquina escolhe a primeira fileira que satisfaz a condição nova quantidade $<$ quantidade original, garantindo que a jogada seja válida.

- Se o *Nim-Sum* for igual a zero, a máquina faz uma jogada aleatória, que é basicamente uma jogada de dificuldade fácil.

**Exemplo de Jogada Estratégica:**
Considere o seguinte exemplo:
- As fileiras de palitos são: `[3, 4, 5]`. - O *Nim-Sum* é calculado como $3 \oplus 4 \oplus 5 = 6$ (em binário, $0011 \oplus 0100 \oplus 0101 = 0110$).

Como o *Nim-Sum* não é zero, a máquina escolhe a fileira a ser modificada. A máquina verifica, por exemplo, que ao remover 2 palitos da fileira 2 (que contém 4 palitos), a configuração do jogo seria `[3, 2, 5]`, com um novo *Nim-Sum* igual a zero:

$$3 \oplus 2 \oplus 5 = 0$$

A máquina faz a jogada removendo 2 palitos da fileira 2, deixando o estado do jogo em `[3, 2, 5]`, e o *Nim-Sum* agora é zero. Isso coloca o jogador em uma posição de desvantagem, pois qualquer jogada que ele fizer pode ser respondida de forma a manter o *Nim-Sum* igual a zero, garantindo a vitória da máquina.

**Cálculo da Jogada:**
A máquina utiliza a seguinte expressão para calcular a quantidade de palitos a ser removida da fileira $i$:

$$maq\_n = filas[i] - (filas[i] \oplus somXor)$$

Aqui, $filas[i]$ é o número de palitos na fileira $i$, e $somXor$ é o *Nim-Sum* das fileiras. O valor de $maq\_n$ representa a quantidade de palitos a ser removida da fileira $i$ para que o novo *Nim-Sum* seja zero.

Por exemplo, considerando as fileiras `[3, 4, 5]` e o *Nim-Sum* igual a 6, a máquina verifica para a fileira 2 (que tem 4 palitos):

$$maq\_n = 4 - (4 \oplus 6) = 4 - 2 = 2$$

Portanto, a máquina remove 2 palitos da fileira 2.

## 4.2 Implementação da Jogada Difícil

A implementação da jogada difícil da máquina no código Haskell é a seguinte:

```
1 -- Jogada dif cil da m quina
2 maquinaDificil :: StateT GameState IO ()
3 maquinaDificil = do
4     (filas, dificuldade) <- get
5     let somXor = calculaXorSom filas  -- Calcula o Nim-Sum
```

```
6       if somXor /= 0 then do
7           let (maq_i, maq_n) = head [(i, filas !! i - (filas !! i `xor`
                somXor)) | i <- [0..length filas - 1], (filas !! i `xor`
                somXor) < filas !! i]
8           let novasFilas = modificarFileira filas maq_i maq_n
9           put (novasFilas, dificuldade)
10          liftIO $ putStrLn $ "M quina fez jogada estrat gica: removeu
                " ++ show maq_n ++ " palitos da fileira " ++ show maq_i
11          if(dificuldade == Dificil_Ajuda) then do
12              liftIO $ putStrLn $ "Nim_sum = " ++ show (calculaXorSom
                    novasFilas)
13          else
14              liftIO $ putStrLn $ "--"
15      else do
16          maquinaFacil  -- Caso o Nim-Sum seja zero, faz uma jogada
                f cil
```

Neste código, a máquina calcula o *Nim-Sum* das fileiras de palitos. Se o *Nim-Sum* não for zero, ela busca uma fileira onde a jogada estratégica pode ser feita. Caso contrário, a máquina realiza uma jogada fácil.

### 4.3 Conclusão

A estratégia vencedora no jogo de Nim é baseada no cálculo do *Nim-Sum* e na modificação das fileiras de palitos de maneira a deixar o adversário em uma posição onde o *Nim-Sum* seja zero após sua jogada. A máquina difícil segue essa estratégia para garantir sua vitória, enquanto a máquina fácil joga de forma aleatória.

## 5 Descrição das Funções

### 5.1 Função `inicializarJogo`

A função `inicializarJogo` é responsável por criar a configuração inicial do jogo, incluindo a lista de fileiras de palitos e o modo de dificuldade escolhido pelo jogador.

- **Entrada:** - Um valor do tipo `Escolha`, que indica a dificuldade ou modo de jogo (Fácil, Difícil, Difícil com Ajuda, ou Máquinas contra Máquinas).

- **Saída:** - Um valor do tipo `State StdGen GameState`, que encapsula o estado do jogo e inclui a lista de fileiras de palitos e a dificuldade escolhida.

- **O que faz:** - A função utiliza um gerador de números aleatórios (do tipo `StdGen`) para gerar a lista de fileiras de palitos. Essa lista é composta por números aleatórios gerados pela função auxiliar `gerarListaAleatoriaState`. Em seguida, ela retorna o estado inicial do jogo, representado como uma tupla com a lista de fileiras e a dificuldade do jogo.

### 5.2 Função `maquinaFacil`

A função `maquinaFacil` implementa a jogada automática da máquina no modo de dificuldade fácil, realizando uma escolha aleatória de fileira e quantidade de palitos para remover.

- **Entrada:** - Nenhuma entrada explícita. A função utiliza o estado atual do jogo (`GameState`), obtido por meio da monada `StateT`.

- **Saída:** - Nenhuma saída explícita. A função modifica o estado do jogo (`GameState`) e imprime no console a jogada realizada pela máquina.

- **O que faz:** - A função realiza os seguintes passos:

- Obtém o estado atual do jogo, que inclui a lista de fileiras de palitos e a dificuldade escolhida.
- Filtra as fileiras com palitos restantes (valores diferentes de zero).
- Escolhe aleatoriamente uma fileira não vazia e uma quantidade de palitos para remover dessa fileira, utilizando a função `randomRIO`.
- Atualiza a lista de fileiras de palitos, removendo a quantidade escolhida da fileira selecionada, por meio da função auxiliar `modificarFileira`.
- Atualiza o estado do jogo com a nova configuração de fileiras.
- Imprime no console a quantidade de palitos removida e a fileira afetada pela jogada da máquina.

## 5.3 Função `maquinaDificil`

A função `maquinaDificil` implementa a jogada automática da máquina no modo de dificuldade difícil. Neste modo, a máquina utiliza a lógica do jogo Nim para realizar jogadas estratégicas com base no valor do Nim-sum (`XOR` das fileiras de palitos).

- **Entrada:** - Nenhuma entrada explícita. A função utiliza o estado atual do jogo (`GameState`), obtido por meio da monada `StateT`.

- **Saída:** - Nenhuma saída explícita. A função modifica o estado do jogo (`GameState`) e imprime no console a jogada estratégica ou, caso não seja possível, uma jogada aleatória.

- **O que faz:** - A função realiza os seguintes passos:

- Obtém o estado atual do jogo, incluindo a lista de fileiras de palitos e a dificuldade escolhida.
- Calcula o valor do `XOR` das fileiras de palitos (`Nim-sum`), utilizando a função auxiliar `calculaXorSom`.
- Se o `Nim-sum` for diferente de zero, realiza uma jogada estratégica:
  - Encontra a fileira (`maq_i`) e o número de palitos a remover (`maq_n`) para reduzir o `Nim-sum` a zero.
  - Atualiza a lista de fileiras de palitos, removendo a quantidade calculada da fileira selecionada, usando a função auxiliar `modificarFileira`.
  - Atualiza o estado do jogo com a nova configuração de fileiras.
  - Imprime no console a jogada estratégica realizada pela máquina.
- Se a dificuldade for `Dificil_Ajuda`, imprime o novo valor do `Nim-sum` após a jogada; caso contrário, imprime uma mensagem padrão.
- Se o `Nim-sum` for zero, delega a jogada para a função `maquinaFacil`, que realiza uma jogada aleatória.

## 5.4 Função `jogoDificil`

A função `jogoDificil` implementa o fluxo do jogo no modo de dificuldade difícil. Neste modo, o jogador e a máquina alternam jogadas, sendo que a máquina realiza uma jogada mais estratégica, dificultando a vitória do jogador. O jogo continua até que todas as fileiras de palitos sejam esvaziadas.

- **Entrada:** - Nenhuma entrada explícita. A função utiliza o estado atual do jogo (`GameState`), obtido por meio da monada `StateT`. - O jogador interage com o jogo através de entradas solicitadas pelo console.

- **Saída:** - Nenhuma saída explícita. A função modifica o estado do jogo (`GameState`) e exibe mensagens no console indicando o progresso e o resultado do jogo.

- **O que faz:** - A função realiza os seguintes passos:

- Chama a função `maquinaDificil`, que faz com que a máquina execute uma jogada estratégica.
- Obtém o estado atual do jogo, incluindo a lista de fileiras de palitos e a dificuldade escolhida.
- Verifica se todas as fileiras de palitos estão vazias:
  - Caso positivo, exibe uma mensagem de derrota (`Você perdeu :(`).
  - Caso contrário, exibe as fileiras de palitos no console utilizando a função auxiliar `imprimirPalitinhos`.
- Solicita ao jogador:
  - A escolha de uma fileira válida (que contenha pelo menos um palito), garantindo isso com a função auxiliar `pedirNumero`.
  - A quantidade de palitos a remover da fileira selecionada, validando se o valor está dentro do intervalo permitido.
- Atualiza a lista de fileiras de palitos com a jogada do jogador, removendo a quantidade de palitos escolhida.
- Verifica novamente se todas as fileiras estão vazias:
  - Caso positivo, exibe uma mensagem de vitória (`Parabéns, você ganhou!`).
  - Caso contrário, chama recursivamente a função `jogoDificil`, continuando o fluxo do jogo.

## 5.5 Função `jogoMaquinaContraMaquina`

A função `jogoMaquinaContraMaquina` implementa o fluxo do jogo entre duas máquinas: uma fácil e outra difícil. As máquinas alternam jogadas até que todas as fileiras de palitos sejam esvaziadas, e o jogo termina quando uma das máquinas vence.

- **Entrada:** - Nenhuma entrada explícita. A função utiliza o estado atual do jogo (`GameState`), obtido por meio da monada `StateT`.

- **Saída:** - Nenhuma saída explícita. A função modifica o estado do jogo (`GameState`) e exibe mensagens no console indicando o progresso e o resultado do jogo.

- **O que faz:** - A função realiza os seguintes passos:

- Obtém o estado atual do jogo, incluindo a lista de fileiras de palitos e a dificuldade escolhida.
- Exibe as fileiras de palitos no console utilizando a função auxiliar `imprimirPalitinhos`.
- Verifica se todas as fileiras estão vazias:
  - Caso positivo, exibe uma mensagem de término do jogo (`O jogo terminou!`).
  - Caso contrário, continua o fluxo do jogo.
- Inicia o turno da máquina fácil:
  - Exibe uma mensagem informando que é o turno da Máquina Fácil.
  - Chama a função `maquinaFacil` para que a máquina fácil realize uma jogada.
  - Verifica se todas as fileiras estão vazias após a jogada da máquina fácil:
    * Caso positivo, exibe a mensagem de vitória da Máquina Fácil (`Máquina Fácil venceu!`).
    * Caso contrário, passa para o turno da máquina difícil.
- Inicia o turno da máquina difícil:
  - Exibe uma mensagem informando que é o turno da Máquina Difícil.
  - Chama a função `maquinaDificil` para que a máquina difícil realize uma jogada.
  - Verifica se todas as fileiras estão vazias após a jogada da máquina difícil:
    * Caso positivo, exibe a mensagem de vitória da Máquina Difícil (`Máquina Difícil venceu!`).
    * Caso contrário, chama recursivamente a função `jogoMaquinaContraMaquina`, continuando o fluxo do jogo.

## 5.6 Função `pedirNumero`

A função `pedirNumero` solicita ao usuário a entrada de um número, validando a entrada com base em um predicado fornecido. Caso a entrada não seja válida, a função solicita novamente até que um valor válido seja informado.

- **Entrada:** - `prompt`: Uma mensagem a ser exibida para o usuário, solicitando a entrada de um número. - `validacao`: Um predicado (função) que recebe um número inteiro e retorna um valor booleano, indicando se a entrada é válida ou não.
- **Saída:** - Retorna um número inteiro (`Int`) que satisfaz o predicado de validação.
- **O que faz:** - A função realiza os seguintes passos:

- Exibe o `prompt` para o usuário, solicitando a entrada de um número.
- Lê a entrada do usuário e tenta convertê-la para um número inteiro usando a função `readMaybe`, que retorna um valor do tipo `Maybe Int`.
- Se a entrada for válida e o número atender ao predicado `validacao`, retorna o número.

- Caso a entrada seja inválida (não seja um número ou não satisfaça a validação), exibe uma mensagem de erro e chama recursivamente a função `pedirNumero` até que uma entrada válida seja fornecida.

- O uso da monada `Maybe` permite tratar de forma segura a possibilidade de falha na conversão de uma entrada, sem gerar erros no programa. Se a conversão falhar, a função continua pedindo uma nova entrada ao usuário.

## 5.7 Função `modificarFileira`

A função `modificarFileira` modifica uma fileira específica no jogo, com base no índice da fileira e na quantidade de palitos a serem removidos. O valor da fileira é atualizado, garantindo que não fique negativo.

- **Entrada:** - `filas`: Uma lista de inteiros representando as fileiras de palitos no jogo. - `i`: O índice da fileira a ser modificada. - `n`: A quantidade de palitos a ser removida da fileira selecionada.

- **Saída:** - Retorna uma nova lista de inteiros representando o estado atualizado das fileiras após a remoção dos palitos.

- **O que faz:** - A função realiza os seguintes passos:

- Utiliza a função `splitAt` para dividir a lista de fileiras em duas partes: a parte antes da fileira a ser modificada (`antes`) e a parte após a fileira (`depois`).

- Subtrai a quantidade de palitos (`n`) da fileira selecionada (`fileira`), garantindo que o valor não fique negativo usando a função `max 0`.

- Junta novamente as partes da lista (`antes` e `depois`), substituindo a fileira modificada pela nova quantidade de palitos.

## 5.8 Função `imprimirPalitinhos`

A função `imprimirPalitinhos` imprime as fileiras de palitos de forma visual no console. Cada fileira é representada por uma linha com a quantidade de palitos correspondente, usando o caractere `|` para cada palito.

- **Entrada:** - `filas`: Uma lista de inteiros representando as fileiras de palitos no jogo, onde cada inteiro indica o número de palitos em uma fileira.

- **Saída:** - Nenhuma saída explícita. A função exibe as fileiras de palitos no console.

- **O que faz:** - A função realiza os seguintes passos:

- Utiliza a função `zip` para associar cada índice da lista `filas` a um número (começando de 0), criando uma lista de pares `(índice, número de palitos)`.

- Para cada par `(i, n)`, onde `i` é o índice da fileira e `n` é o número de palitos, imprime uma linha no formato `i: |...|`, onde o número de `|` corresponde ao valor de `n`.

- Utiliza a função `replicate` para gerar a sequência de `|` com o número de repetições igual a `n`.

## 5.9 Função `gerarListaAleatoriaState`

A função `gerarListaAleatoriaState` gera uma lista aleatória de números inteiros, representando as fileiras de palitos no jogo. O número de fileiras é aleatório, e as quantidades de palitos em cada fileira são números ímpares gerados aleatoriamente dentro de um intervalo.

- **Entrada:** - Nenhuma entrada explícita. A função utiliza o estado atual do gerador de números aleatórios (`StdGen`), que é acessado por meio da monada `State`.

- **Saída:** - Retorna uma lista de inteiros representando as fileiras de palitos, gerada aleatoriamente.

- **O que faz:** - A função realiza os seguintes passos:

- Obtém o estado atual do gerador de números aleatórios (`g`) usando a monada `State`.
- Gera aleatoriamente um número (`num`) entre 2 e 5, que determinará o número de fileiras de palitos.
- Utiliza a função `randomRs` para gerar uma lista infinita de números aleatórios entre 1 e 7, e então filtra apenas os números ímpares.
- Toma os primeiros `num` números da lista filtrada, formando a lista de fileiras de palitos.
- Atualiza o estado do gerador de números aleatórios com o novo estado (`g'`).
- Retorna a lista gerada de fileiras de palitos.

## 5.10 Função `calculaXorSom`

A função `calculaXorSom` calcula a soma XOR de uma lista de números inteiros. A operação XOR é aplicada sucessivamente sobre os elementos da lista, começando com o valor 0.

- **Entrada:** - `[Int]`: Uma lista de inteiros sobre os quais a operação XOR será aplicada.

- **Saída:** - Retorna um número inteiro (`Int`) que é o resultado da soma XOR de todos os elementos da lista.

- **O que faz:** - A função realiza os seguintes passos:

- Utiliza a função `foldl` para aplicar a operação `xor` de forma acumulativa sobre a lista.
- Inicia a operação XOR com o valor 0.
- Para cada elemento `e` da lista, aplica a operação `xor` entre o acumulador e o elemento, e atualiza o acumulador com o resultado.
- Retorna o valor final do acumulador, que é a soma XOR de todos os elementos da lista.

## 5.11 Função `escolherDificuldade`

A função `escolherDificuldade` solicita ao usuário a escolha da dificuldade do jogo e a retorna. Caso o usuário forneça uma entrada inválida, a função solicita uma nova escolha até que uma entrada válida seja fornecida.

- **Entrada:** - Nenhuma entrada explícita. A função lê a entrada do usuário a partir do console.

- **Saída:** - Retorna um valor do tipo `Escolha`, que representa a dificuldade escolhida pelo usuário.

- **O que faz:** - A função realiza os seguintes passos:

- Exibe uma mensagem solicitando ao usuário que escolha a dificuldade do jogo. As opções são: `Facil`, `Dificil`, `Dificil_Ajuda`, `MvsM` e `Sair`.
- Lê a escolha do usuário usando `getLine`.
- Tenta converter a entrada para o tipo `Escolha` usando a função `readMaybe`.
- Se a conversão for bem-sucedida, retorna o valor da dificuldade escolhida.
- Se a conversão falhar (quando o usuário fornece uma entrada inválida), exibe uma mensagem de erro e chama recursivamente a função `escolherDificuldade` para que o usuário possa tentar novamente.

## 5.12 Função `main`

A função `main` é a função principal do programa, responsável por iniciar o jogo, solicitar a escolha da dificuldade ao usuário e gerenciar o fluxo de execução do jogo com base na escolha feita. Ela também reinicia o jogo após cada partida, permitindo múltiplas rodadas até que o jogador decida sair.

- **Entrada:** - Nenhuma entrada explícita. A função interage com o usuário por meio do console, solicitando sua escolha de dificuldade.

- **Saída:** - Nenhuma saída explícita. A função executa o jogo de acordo com a dificuldade escolhida, exibindo informações e resultados no console.

- **O que faz:** - A função realiza os seguintes passos:

- Exibe a mensagem inicial `"Jogo dos Palitos"`.
- Solicita ao usuário que escolha a dificuldade por meio da função `escolherDificuldade`.
- Obtém um novo gerador de números aleatórios (`g`) usando `newStdGen`.
- Inicializa o estado do jogo com a função `inicializarJogo`, utilizando a escolha do usuário e o gerador de números aleatórios.
- Dependendo da escolha do usuário (`Facil`, `Dificil`, `Dificil_Ajuda`, `MvsM` ou `Sair`), a função executa uma das opções de jogo:
  - Se a escolha for `Facil`, o jogo fácil é iniciado com a função `jogoFacil`.
  - Se a escolha for `Dificil`, o jogo difícil é iniciado com a função `jogoDificil`.
  - Se a escolha for `Dificil_Ajuda`, o jogo difícil com ajuda é iniciado com a função `jogoDificil`.

  - Se a escolha for `MvsM`, um jogo entre máquinas é iniciado com a função `jogoMaquinaContraMaquina`.
  - Se a escolha for `Sair`, o programa exibe a mensagem `"Até mais!"` e encerra o jogo.

- Após o término de cada partida, a função `main` é chamada recursivamente, permitindo que o jogador escolha novamente a dificuldade ou saia do jogo.

# 6 Considerações Finais

O trabalho implementa eficientemente o Jogo dos Palitos, utilizando conceitos fundamentais de programação funcional e estratégia matemática. A separação clara das funções facilita a manutenção e possíveis extensões do projeto, como a adição de novas dificuldades ou modos de jogo. A estratégia vencedora demonstra como princípios matemáticos podem ser aplicados para resolver problemas de jogos combinatórios.

# Código-Fonte

O código completo está disponível no repositório oficial: `https://github.com/ArturMeyer/jogo-dos-palitos-haskell.git`.