FUNDAÇÃO EDSON QUEIROZ

UNIVERSIDADE DE FORTALEZA - UNIFOR

CENTRO DE CIÊNCIAS TECNOLÓGICAS

CURSO ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Programação Funcional - Pedra, Papel e Tesoura

João Pedro Andrade Oliveira - 2222943

Yago Gomes Varela - 2222852

Kauan dos Anjos Meneses - 2222840

Luana Fernanda de Sousa Gomes - 2213811

Alex Custodio Rabelo Gomes - 2222900

Francisco Rodrigues Barbosa Neto - 2222863

Fortaleza/Ceará

2024

Introdução

O presente trabalho tem como objetivo desenvolver e analisar um jogo de "Pedra, Papel e Tesoura" implementado em linguagem JavaScript, destacando a aplicação dos conceitos de programação funcional e boas práticas de desenvolvimento. O código foi estruturado para permitir a interação entre dois jogadores, sendo um deles controlado pelo usuário e o outro pelo sistema com jogadas aleatórias. Além de cumprir os requisitos funcionais, como a escolha das jogadas, a validação de entradas e a atualização do placar, o projeto também foca em requisitos não funcionais, como usabilidade, desempenho, confiabilidade e manutenção, assegurando uma experiência fluida e eficiente para o jogador.

Definição de papéis

Alex: responsável por criar o relatório.

Francisco: responsável por criar requisitos funcionais.

João: responsável por criar requisitos não funcionais.

Kauan: responsável por criar o repositório no github e subir pra vercel, html e css.

Luana: responsável por criar o código.

Yago: responsável por fazer testes.

Requisitos Funcionais e Não Funcionais

Requisitos funcionais:

1. Escolher jogada (Jogador 1)
    a. O jogador 1 faz sua escolha através da função jogar(), que obtém o valor do campo de entrada HTML document.getElementById('escolhaJogador1').value.

2. Escolha aleatória para Jogador 2
    a. O jogador 2 tem sua escolha feita aleatoriamente na linha: const jogador2Escolha = listaJogadas[Math.floor(Math.random() * listaJogadas.length)].

3. Decidir o vencedor
    a. A função decideGanhador(jogador1, jogador2, regraVencedor) é responsável por determinar o vencedor com base nas regras.

4. Atualizar e exibir o placar
    a. O placar é atualizado pela função atualizarPlacar(vencedor) e exibido no HTML com document.getElementById('placar').textContent = ${jogador1Vitorias} : ${jogador2Vitorias}``.

5. Exibir o resultado da rodada
    a. O resultado da rodada é mostrado com document.getElementById('resultado').innerHTML = ${vencedor}``.

6. Validação de entrada
    a. A validação da jogada do jogador 1 ocorre quando o código verifica se a entrada é válida na linha: if (!listaJogadas.includes(jogador1Escolha)), seguida de um alerta em caso de erro.

Requisitos não funcionais:

1. Usabilidade
    a. A interface interage com o usuário mostrando mensagens claras como a validação de entrada e exibindo os resultados e o placar em tempo real. Isso melhora a experiência do usuário.
2. Desempenho
    a. O código é eficiente com processamento leve, utilizando funções simples e rápidas para calcular os resultados e atualizar o placar.
3. Extensibilidade
    a. O código permite fácil adição de novas jogadas (como "lagarto" e "Spock", por exemplo), pois as regras e as jogadas estão separadas e organizadas de forma modular.
4. Manutenibilidade
    a. A estrutura modular, como as funções separadas (decideGanhador, criarPlacar), facilita a manutenção e a atualização do código.
5. Confiabilidade

a. O código usa uma validação para garantir que o jogador 1 faça uma jogada válida e só atualiza o placar corretamente após a decisão do vencedor, garantindo resultados confiáveis.

6. Feedback imediato
    a. O feedback é fornecido imediatamente após a jogada, exibindo o resultado da rodada e o placar atualizado na interface, conforme document.getElementById('resultado') e document.getElementById('placar').

Construções funcionais

List Comprehension

List Comprehension sendo usado o map() para criar uma nova lista de jogadas.

```
// List Comprehension sendo usado o map() para criar uma nova lista de jogadas.
const listaJogadas = jogadas.map(jogada => jogada)
```

Função de alta ordem

Função de alta ordem que está recebendo como parâmetro regraVencedor e a utilizando para decidir o vencedor.

```
// Função de Auta ordem que esta recebendo como parametro regraVencedor e a ultilizando para decidir o vencedor
function decideGanhador(jogador1, jogador2, regraVencedor) {
    return regraVencedor(jogador1, jogador2)
}
```

Função lambda

Função lambda que está sendo usada na forma anônima (arrow function) na função decideGanhador.

```
// Função lambda que esta sendo usada de forma anonima (arrowfunction) na função decideGanhador.
const regra = (jogador1, jogador2) => {
    if (jogador1 === jogador2) return "Ninguém ganhou! Deu empate :)"
    if ((jogador1 === "pedra" && jogador2 === "tesoura") ||
        (jogador1 === "tesoura" && jogador2 === "papel") ||
        (jogador1 === "papel" && jogador2 === "pedra")) {
            return "Parabéns! Jogador 1 venceu essa rodada!!!"
        } else {
            return "Parabéns! Jogador 2 venceu essa rodada!!!"
        }
}
```

Closure

criarPlacar está sendo criado e retornando uma closure. Closure em uma função interna que tem acesso às variáveis externas.

```
// Função que está criando e retornando umna closure
function criarPlacar() {
    let jogador1Vitorias = 0
    let jogador2Vitorias = 0
    // Closure uma função interna que tem acesso as variaveis externas
    return function(vencedor) {
        if (vencedor === "Parabéns! Jogador 1 venceu essa rodada!!!") {
            jogador1Vitorias++
        } else if (vencedor === "Parabéns! Jogador 2 venceu essa rodada!!!") {
            jogador2Vitorias++
        }
        return {jogador1Vitorias, jogador2Vitorias}
    }
}
```

Chatbot

O código foi criado com chatbot, mas adaptamos as variáveis e deixamos mais humano e simpático. Também pedimos ajuda ao chat de como poderíamos fazer o teste com o jest: https://chatgpt.com/share/66e83941-9a4c-8006-bf4f-e83e08f4ad94.

Código fonte no github

https://github.com/KauanAnjos/Atividade-de-programacao-funcional