| **COMPONENTES DO GRUPO** | |
| --- | --- |
| ***ID*** | ***Nome do Aluno*** |
| 1 | Isaac |
| 2 | Ezequiel |
| 3 | Lucas |
| 4 | Haller |

| **CONTROLE DE VERSÃO** | | |
| --- | --- | --- |
| ***ID*** | ***Descrição da Versão*** | ***Data da Versão*** |
| 1 | Versão 1 | 14/09/2024 |

| **Diagrama de Classes** |
| --- |

### **Classe GameService**

Responsável por gerenciar a lógica central do jogo. Atua como o controlador principal que coordena o tabuleiro, os jogadores e a lógica de vitória ou empate.

* **Atributos:**
  + tabuleiro: Signal<EstadoCelula[][]>: matriz que representa o estado atual do tabuleiro, onde cada célula pode estar vazia ou ocupada por um dos jogadores. Torna o estado do tabuleiro reativo, disparando atualizações automáticas em outras partes do sistema quando alterado.
  + jogadores:Jogador[]: array contendo os jogadores que estão participando do jogo.
  + jogadorAtual: Signal<Jogador>: representa o jogador que tem a próxima jogada, indicando mudanças que podem ser monitoradas por outros componentes.
* **Métodos:**
  + constructor(): inicializa a instância da classe, configurando o tabuleiro, os jogadores e definindo o primeiro jogador a jogar.
  + Jogar(coluna: number): void: executa uma jogada na coluna especificada e coordena a jogada, alternando o jogador atual e verificando se há uma condição de vitória ou empate.
  + obterLinhaDisponivel(coluna: number): number: retorna a primeira posição (linha) disponível na coluna especificada onde uma peça pode ser colocada.
  + marcarCelula(coluna: number, linha: number): void: marca uma posição específica no tabuleiro como ocupada pelo jogador atual.
  + alternarJogador(): void: alterna o controle para o próximo jogador.
  + verificarVitoria(coluna: number, linha: number): boolean: verifica se a jogada mais recente resultou em vitória, verificando as várias direções (horizontal, vertical, diagonal).
  + verificarEmpate(): boolean: verifica se o jogo terminou em empate, ou seja, se o tabuleiro está cheio sem uma condição de vitória.
  + Métodos de verificação: (verificarHorizontal, verificarVertical, verificarDiagonal, verificarDiagonalPrincipal, verificarDiagonalSecundaria): métodos que verificam as diferentes direções no tabuleiro para determinar se há uma sequência de células que constitui uma vitória.
  + ReiniciarJogo(): void: reseta o jogo, limpando o tabuleiro e reconfigurando os jogadores.

### 

### 

### 

### **Classe Jogador**

Representa os participantes do jogo.

* **Atributos:**
  + id: number: identificador único para cada jogador.
  + nome: string: nome do jogador.
  + cor: string: cor associada ao jogador, provavelmente usada para identificar visualmente as peças no tabuleiro.
  + pontuacao: number: pontuação acumulada do jogador.
  + venceu: boolean: indicador de se o jogador venceu ou não o jogo atual.
  + imagemVitoria: string: URL ou caminho para a imagem que será exibida quando o jogador vencer.

### **Enum EstadoCelula**

Define os possíveis estados de uma célula no tabuleiro.

* **Valores:**
  + Vazio: indica que a célula está vazia e disponível para jogar.
  + Jogador1: indica que a célula está ocupada pelo primeiro jogador.
  + Jogador2: indica que a célula está ocupada pelo segundo jogador.

### **Classe TabuleiroComponent**

Representa a interface gráfica (UI) do tabuleiro, sendo responsável por renderizar o estado do jogo e interagir com o usuário.

* **Atributos:**
  + tabuleiro: Signal<EstadoCelula[][]>: referência ao tabuleiro do jogo, que deve ser sincronizado com o estado gerido pelo GameService.
  + jogadorAtual: Signal<Jogador>: jogador atual, sincronizado com o GameService.
  + esperando: boolean: indica se o componente está aguardando a jogada de um jogador.
  + vencedor: boolean: indica se há um vencedor no jogo atual.
  + jogadorVencedor: Jogador: armazena o jogador vencedor, se houver.
  + celulasAtivas: boolean[][]: matriz de booleanos indicando quais células estão ativas e disponíveis para jogadas.
* **Métodos:**
  + Constructor(gameService: GameService): recebe uma instância de GameService, conectando a lógica do serviço à interface de usuário.
  + Jogar(coluna: number): void: responsável por capturar a ação do usuário de jogar em uma coluna específica e delegar essa ação ao GameService.
  + getCorJogador(celula: EstadoCelula): string: retorna a cor associada ao jogador que ocupa uma célula específica, usada para renderizar a célula corretamente na interface.

### **Relacionamentos entre as Classes**

* **GameService e TabuleiroComponent**: TabuleiroComponent depende de GameService para funcionar corretamente. GameService mantém a lógica do jogo e o estado do tabuleiro, enquanto TabuleiroComponent lida com a interface e interação do usuário.
* **GameService e Jogador**: GameService gerencia a alternância e a verificação de vitórias entre os jogadores.
* **Signal**: utilizado para permitir reatividade, de modo que mudanças no estado (como o tabuleiro ou o jogador atual) possam ser observadas e atualizadas automaticamente em outras partes do sistema.

Esta arquitetura procura manter a separação entre a lógica do jogo e a interface de usuário, visando a facilitar tanto a manutenção quanto a evolução e escalabilidade do código.