Enunciado do Trabalho de Programação Java: Implementação do Jogo Batalha Naval

Objetivo:
Desenvolver um programa em Java que simule o jogo "Batalha Naval". O programa deve implementar todas as funcionalidades principais do jogo utilizando métodos bem definidos para organizar as ações e regras. O objetivo é reforçar conceitos de programação estruturada, manipulação de arrays e lógica de jogo.

Descrição do Jogo:
O jogo consiste em dois jogadores (Jogador 1 e Jogador 2), cada um com um cenário (tabuleiro) de tamanho 10 x10. Em cada cenário, devem ser posicionadas 5 embarcações de tamanhos distintos:

- **1 unidade (1 embarcação)**
- **2 unidades (1 embarcação)**
- **3 unidades (1 embarcação)**
- **4 unidades (1 embarcação)**
- **5 unidades (1 embarcação)**

Os jogadores atacam alternadamente, e o objetivo é afundar todas as embarcações do adversário. O programa deve verificar se o ataque acertou ou errou e atualizar o tabuleiro adequadamente. O jogo termina quando um dos jogadores não possui mais embarcações no tabuleiro.

Funcionalidades e Métodos a Implementar:

1. **Gerar cenários aleatórios para os jogadores:**
    - `public static char[][] gerarCenarioAleatorio()`
        - Gera um tabuleiro 10x10 preenchido com água ('~') e posiciona aleatoriamente as embarcações de tamanhos definidos. As embarcações não podem se sobrepor.
2. **Exibir o cenário do jogador (com embarcações ocultas ou visíveis):**
    - `public static void exibirCenario(char[][] cenario, boolean ocultarEmbarcacoes)`
        - Exibe o tabuleiro na tela. Se `ocultarEmbarcacoes` for `true`, as embarcações devem ser representadas como água ('~').
3. **Realizar um ataque no adversário:**
    - `public static boolean realizarAtaque(char[][] cenarioAdversario, int linha, int coluna)`
        - Registra o ataque na posição especificada. Retorna `true` se o ataque acertar uma embarcação e `false` se for água.

4. **Verificar se uma embarcação foi completamente afundada:**
    - `public static boolean verificarEmbarcacaoAfundada(char[][] cenario, int tamanhoEmbarcacao)`
        - Verifica se todas as partes de uma embarcação de tamanho específico foram destruídas.
5. **Atualizar o cenário com os ataques realizados:**
    - `public static void atualizarCenario(char[][] cenario, int linha, int coluna, boolean acerto)`
        - Atualiza o tabuleiro, marcando 'X' para acerto e 'O' para erro.
6. **Registrar os ataques já realizados:**
    - `public static boolean verificarAtaqueRepetido(boolean[][] ataques, int linha, int coluna)`
        - Verifica se uma posição já foi atacada. Retorna `true` se o ataque for repetido.
7. **Exibir a quantidade de embarcações afundadas:**
    - `public static int contarEmbarcacoesAfundadas(char[][] cenario)`
        - Retorna o número de embarcações que foram completamente destruídas.
8. **Verificar se um jogador venceu:**
    - `public static boolean verificarVitoria(char[][] cenario)`
        - Retorna `true` se todas as embarcações do adversário forem destruídas.

Requisitos Adicionais:

- O programa deve permitir que o jogador jogue contra outro jogador, contra a máquina (modo automático) ou máquina contra máquina (modo automático).
- A inteligência da máquina deve atacar posições aleatórias.
- Usar constantes para representar os elementos do tabuleiro (ex.: `final char AGUA = '~'; final char NAVIO = 'N'; final char ACERTO = 'X'; final char ERRO = 'O';`).
- Ao final gere um relatório com as ações de ambos os jogadores.
- O programa deve ser modular, com cada funcionalidade implementada em métodos separados.
- Documente o código com comentários explicativos sobre as funcionalidades dos métodos.

Entrega:

- Data entrega 10/01/2025 até as 23h59, via SIGAA
- Individual ou dupla
- Os métodos sugeridos neste enunciado são mínimos, ficando a cargo do aluno acrescentar outros se necessário
- Submeter o código-fonte do programa em um arquivo `.java`.
- Um documento README explicando como o programa funciona e os testes realizados.

Critérios de Avaliação:

1. Implementação correta de todas as funcionalidades e métodos.
2. Organização e clareza do código.
3. Tratamento de erros (ex.: evitar ataques fora do tabuleiro ou repetidos).
4. Testes realizados para verificar o funcionamento completo do jogo.
5. Documentação do código.

Boa sorte e divirta-se programando!