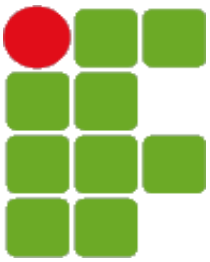

Instituição: Instituto Federal de Santa Catarina (IFSC)
Disciplina: Pensamento Computacional e Algoritmos
Professor: Sergio Mauricio Prolo Santos Junior
Alunos: Gustavo Ramos Rocha, Ana Luiza Mattia de Lima.

Relatório

Geração e Validação de Tabuleiros de Batalha Naval

Introdução

A atividade realizada tem como objetivo estar construindo um algoritmo de campo de Batalha Naval em Java. Nosso desafio foi gerar campos aleatórios válidos e receber possíveis campos pelo shell e informar no mesmo se o campo inserido é válido ou não. Quando o algoritmo receber a instrução ‘G’ ele irá gerar um Campo de Batalha Naval 10x10 coberto de água ‘.’ e irá posicionar os 5 navios de forma válida no campo. Quando o algoritmo receber a instrução V, ele receberá um arquivo por redirecionamento de entrada no Shell e irá informar se o Campo de Batalha é válido ou não.

Definição do Problema

No desenvolvimento do projeto e definição do campo de batalha naval, para ele ser válido, precisa seguir algumas regras pré-definidas:

- Deve ser composto de uma matriz 10x10.
- Possui somente os símbolos dos 5 barcos e de água ('P','E','C','S','N').
- Cada tipo de navio deve aparecer exatamente uma vez no tabuleiro.
- Barcos devem estar posicionados na matriz respeitando os tamanhos respectivos de cada.

- Os barcos devem estar posicionados somente na horizontal ou na vertical.

Tabela 1: Navios utilizados no tabuleiro.

Navio	Tamanho	Símbolo
Porta-aviões	5	P
Encouraçado	4	E
Cruzador	3	C
Submarino	3	S
Contratorpedeiro	2	N

Modos de Operação

O programa opera em dois modos distintos, definidos por argumentos da linha de comando:

Geração (G)

Quando executado com o argumento ‘G’, o programa gera um tabuleiro 10x10 vazio, posiciona aleatoriamente um navio de cada tipo, garante que todos os navios estejam completamente dentro do tabuleiro, impede sobreposição entre navios e por fim imprime o tabuleiro na saída padrão.

Validação (V)

Quando executado com o argumento ‘V’, o programa lê um tabuleiro via redirecionamento de entrada, verifica se as dimensões são exatamente 10×10, valida se todos os símbolos são reconhecidos (P, E, C, S, N, ou .), confirma a presença de exatamente um navio de cada tipo, verifica se cada navio possui o tamanho correto, garante que cada navio está totalmente horizontal ou vertical, informa se o tabuleiro é válido ou apresenta o erro específico encontrado.

Nesse modo os tipos de saída que podem ocorrer, são:

- Tabuleiro válido: Tabuleiro VÁLIDO
- Dimensões incorretas: Tabuleiro INVÁLIDO: dimensões incorretas

- Símbolo desconhecido: Tabuleiro INVÁLIDO: navio desconhecido, representado pela letra ‘X’ (utilizando ‘X’ apenas para exemplificar)
- Tamanho incorreto: Tabuleiro INVÁLIDO: navio ‘X’ com tamanho incorreto
- Navio não encontrado: Tabuleiro INVÁLIDO: navio ‘X’ não encontrado
- Múltiplos navios: Tabuleiro INVÁLIDO: múltiplos navios do tipo 'X'
- Navio não linear: Tabuleiro INVÁLIDO: navio ‘X’ não está totalmente na horizontal ou vertical

Caso não tenha sido passado nenhum argumento, o retorno será assim:

```
Uso: java BatalhaNaval <modo>
  G - Gerar tabuleiro aleatório
  V - Validar tabuleiro lido da entrada
```

Implementação

O programa utiliza constantes estáticas (static final) para armazenar informações fixas do jogo, como tamanhos e símbolos dos navios. O tabuleiro é representado por uma matriz bidimensional de caracteres (char[][]), permitindo acesso direto a qualquer posição. Para a geração aleatória, foi utilizada a classe Random do Java, que garante distribuição uniforme no posicionamento dos navios.

O algoritmo de geração funciona através de tentativa e erro: gera posições e orientações aleatórias, verifica se o navio cabe no tabuleiro e se as posições estão livres, e então posiciona o navio. Caso contrário, tenta novamente. Esta abordagem garante que eventualmente todos os navios sejam posicionados de forma válida. A validação ocorre em três etapas sequenciais: primeiro verifica as dimensões do tabuleiro (10×10), depois confirma que todos os símbolos são válidos (P, E, C, S, N ou '.'), e por fim analisa a estrutura dos navios, verificando tamanho, orientação e quantidade. Para evitar contar o mesmo navio múltiplas vezes, utilizamos uma matriz booleana visitado[][].

O método contarNavioContínuo é responsável por identificar e validar cada navio individualmente. Ele verifica os quatro vizinhos imediatos de cada célula e determina se o navio está horizontal, vertical ou isolado. Se detectar vizinhos tanto na horizontal quanto na vertical, identifica corretamente um navio em formato L ou diagonal, reportando erro. O algoritmo percorre o navio em ambas as direções do eixo identificado, marcando todas as células como visitadas e retornando o tamanho total, garantindo contagem precisa independente do ponto inicial.

O código foi estruturado seguindo o princípio de separação de responsabilidades, onde cada método possui uma função única. Os métodos gerarTabuleiro e validarTabuleiro orquestram seus respectivos processos, delegando tarefas específicas para métodos auxiliares como posicionaNavio, verificaPosicao, validarSimbolo e validarTamanhoNavios. Esta organização facilita manutenção, depuração e compreensão do código. O tratamento de erros utiliza System.exit(0) para encerrar imediatamente ao detectar problemas, garantindo que apenas um erro seja reportado por vez com mensagem clara e específica.

Representação do Algoritmo em Pseudocódigo

Método main

```
main (recebe: argumentos da linha de comando)
    Criar scanner para leitura de entrada

    Se não houver argumentos, então
        Exibir instruções de uso
        Encerrar programa

    Converter primeiro argumento para maiúscula e guardar como 'modo'

    Se modo = 'G', então
        Chamar gerarTabuleiro()
    Senão, se modo = 'V', então
        Chamar validarTabuleiro(scanner)
        Exibir "Tabuleiro VÁLIDO"
    Senão
        Exibir "Modo inválido"
        Encerrar programa
```

```
Fechar scanner
```

Método gerarTabuleiro

```
gerarTabuleiro ()
    Criar matriz tabuleiro 10×10

    Para cada linha = 0 até 9
        Para cada coluna = 0 até 9
            Preencher tabuleiro[linha][coluna] com AGUA ('.')

    Criar gerador de números aleatórios

    Para cada índice i = 0 até 4
        Chamar posicionaNavio(tabuleiro, NAVIO[i], TAMANHO_NAVIO[i], random)

    Chamar imprimirTabuleiro(tabuleiro)
```

Método posicionaNavio

```
posicionaNavio (recebe: tabuleiro, simbolo, tamanho, random)
    posicionado = falso

    Enquanto não posicionado
        horizontal = gerar booleano aleatório
        linha = gerar número aleatório de 0 a 9
        coluna = gerar número aleatório de 0 a 9

        Se verificaPosicao(tabuleiro, linha, coluna, tamanho, horizontal), então
            Se horizontal, então
                Para j = coluna até (coluna + tamanho - 1)
                    tabuleiro[linha][j] = simbolo
            Senão
                Para i = linha até (linha + tamanho - 1)
                    tabuleiro[i][coluna] = simbolo

            posicionado = verdadeiro
```

Método verificaPosicao

```
verificaPosicao (recebe: tabuleiro, linha, coluna, tamanho, horizontal)
    Se horizontal, então
        Se (coluna + tamanho) > 10, então
            Retornar falso

        Para j = coluna até (coluna + tamanho - 1)
            Se tabuleiro[linha][j] ≠ AGUA, então
                Retornar falso

    Senão (vertical)
        Se (linha + tamanho) > 10, então
            Retornar falso

        Para i = linha até (linha + tamanho - 1)
            Se tabuleiro[i][coluna] ≠ AGUA, então
                Retornar falso

    Retornar verdadeiro
```

Método imprimirTabuleiro

```
imprimirTabuleiro (recebe: tabuleiro)
    Para cada linha i = 0 até 9
        Para cada coluna j = 0 até 9
            Imprimir tabuleiro[i][j]
            Se j < 9, então
                Imprimir espaço
        Imprimir quebra de linha
```

Método validarTabuleiro

```
validarTabuleiro (recebe: scanner)
    Criar matriz tabuleiroRecebido 10×10

    Para cada linha i = 0 até 9
        Se não houver próxima linha, então
            Exibir "Tabuleiro INVÁLIDO: dimensões incorretas"
            Encerrar programa

        Ler próxima linha e dividir por espaços
```

```
        Se número de partes ≠ 10, então
            Exibir "Tabuleiro INVÁLIDO: dimensões incorretas"
            Encerrar programa

        Para cada coluna j = 0 até 9
            tabuleiroRecebido[i][j] = primeiro caractere da parte[j]

    Se ainda houver linhas, então
        Exibir "Tabuleiro INVÁLIDO: dimensões incorretas"
        Encerrar programa

    Chamar validarSimbolo(tabuleiroRecebido)
    Chamar validarTamanhoNavios(tabuleiroRecebido)
```

Método validarSimbolo

```
validarSimbolo (recebe: tabuleiro)
    Para cada linha i = 0 até 9
        Para cada coluna j = 0 até 9
            simbolo = tabuleiro[i][j]

            Se simbolo ≠ 'P' E simbolo ≠ 'E' E simbolo ≠ 'C' E
               simbolo ≠ 'S' E simbolo ≠ 'N' E simbolo ≠ '.', então
                Exibir "Tabuleiro INVÁLIDO: navio desconhecido, representado pela
letra" + simbolo
                Encerrar programa
```

Método validarTamanhoNavios

```
validarTamanhoNavios (recebe: tabuleiro)
    Criar matriz visitado 10×10 (inicializada com falso)
    Criar arrays qtdNavios[5] e somaTamanhos[5] (inicializados com 0)

    Para cada linha i = 0 até 9
        Para cada coluna j = 0 até 9
            Se tabuleiro[i][j] ≠ AGUA E não visitado[i][j], então
                tipo = tabuleiro[i][j]
                indice = pegaIndiceNavio(tipo)

                Se indice = -1, continuar para próxima iteração
```

```
            tamanho = contarNavioContínuo(tabuleiro, i, j, tipo, visitado)
            qtdNavios[indice]++

            Se tamanho ≠ TAMANHO_NAVIO[indice], então
               Exibir "Tabuleiro INVÁLIDO: navio com tamanho incorreto"
               Encerrar programa

            somaTamanhos[indice] += tamanho

   Para cada índice k = 0 até 4
      Se qtdNavios[k] ≠ 1, então
         Se qtdNavios[k] = 0, então
            Exibir "Tabuleiro INVÁLIDO: navio não encontrado"
         Senão
            Exibir "Tabuleiro INVÁLIDO: múltiplos navios do tipo"
         Encerrar programa
```

Método contarNavioContínuo

```
contarNavioContínuo (recebe: tabuleiro, linha, coluna, tipo, visitado)
   tamanho = 0

   Verificar se há vizinhos do mesmo tipo:
      temDireita = (coluna+1 < 10 E tabuleiro[linha][coluna+1] = tipo)
      temEsquerda = (coluna-1 ≥ 0 E tabuleiro[linha][coluna-1] = tipo)
      temBaixo = (linha+1 < 10 E tabuleiro[linha+1][coluna] = tipo)
      temCima = (linha-1 ≥ 0 E tabuleiro[linha-1][coluna] = tipo)

   Se (temDireita OU temEsquerda) E (temBaixo OU temCima), então
      Exibir "Tabuleiro INVÁLIDO: navio não está totalmente na horizontal ou
vertical"
      Encerrar programa

   Se temDireita OU temEsquerda, então (navio horizontal)
      col = coluna
      Enquanto col < 10 E tabuleiro[linha][col] = tipo
         visitado[linha][col] = verdadeiro
         tamanho++
         col++

      col = coluna - 1
      Enquanto col ≥ 0 E tabuleiro[linha][col] = tipo
         visitado[linha][col] = verdadeiro
         tamanho++
         col--
```

```
    Retornar tamanho

  Se temBaixo OU temCima, então (navio vertical)
    lin = linha
    Enquanto lin < 10 E tabuleiro[lin][coluna] = tipo
      visitado[lin][coluna] = verdadeiro
      tamanho++
      lin++

    lin = linha - 1
    Enquanto lin ≥ 0 E tabuleiro[lin][coluna] = tipo
      visitado[lin][coluna] = verdadeiro
      tamanho++
      lin--

    Retornar tamanho

  (célula isolada - navio de tamanho 1)
  visitado[linha][coluna] = verdadeiro
  Retornar 1
```

Método pegaIndiceNavio

```
pegaIndiceNavio (recebe: tipo)
  Para cada índice k = 0 até 4
    Se NAVIO[k] = tipo, então
      Retornar k

  Retornar -1
```

Conclusão

O projeto foi implementado com sucesso, atendendo aos requisitos funcionais especificados. O programa é capaz de gerar tabuleiros válidos aleatoriamente e validar tabuleiros fornecidos. Durante os testes realizados, o programa retornou resultados corretos para todos os casos testados, incluindo tabuleiros válidos, tabuleiros com dimensões incorretas, símbolos inválidos, navios com tamanho errado, navios faltando, navios duplicados e navios mal posicionados.

Uma observação importante é que, embora o programa identifique se o tabuleiro é válido ou não, no modo V, as mensagens de saída não foram implementadas exatamente como especificado no arquivo de orientação do projeto, não detalhando especificamente quando o navio sofre sobreposição ou navios na diagonal, mas ele considera o tabuleiro ‘Inválido’ nessas situações.

A experiência de desenvolvimento permitiu aplicar na prática conceitos fundamentais como manipulação de matrizes bidimensionais, implementação de algoritmos de busca e validação, geração de números aleatórios e tratamento de casos especiais. A organização do código em métodos com responsabilidades bem definidas facilitou bastante a depuração e os testes durante o desenvolvimento.