Trabalho Prático 01 - Batalha Naval

Objetivo: Desenvolver um mini-jogo de batalha naval. Neste jogo, os jogadores Humano e Computador disparam "tiros" e o resultado da jogada é informado na tela. O jogo termina quando toda a frota de um dos jogadores é afundada. **Valor:** 20 pontos.

Entrega: Ao final desta prática, você deverá enviar no **SIGAA** da disciplina um arquivo **.zip** contendo os arquivos de extensão **.c** e com extensão **.exe** (resultados das compilações) **E** a Documentação (Seção 4) em **.pdf**. O trabalho pode ser feito em grupo de até **três integrantes**. **Atenção:** Apenas um integrante deve submeter o trabalho.

Formato da Entrega:
Defina o nome do arquivo .zip como TP_01_Nome1_Nome2_Nome3.zip.
Atenção: SOMENTE serão aceitos submissões com extensão .zip

Seu código deve ser desenvolvido em C, estar bem organizado e indentado, possuir funções ou procedimentos (se necessário) e conter comentários relevantes. Cópias de trabalho são inaceitáveis, podem ser detectadas automaticamente e serão repassadas ao conselho disciplinar.

1 *Gameplay*

Batalha Naval é um jogo de tabuleiro cujo objetivo é afundar a frota de navios do inimigo. Inicialmente é definido um tabuleiro (matriz) de 10 x 10, os tabuleiros não são visíveis entre os jogadores. Cada jogador usa dois tabuleiros. Antes de iniciar o jogo os jogadores posicionam a sua frota em seu primeiro tabuleiro, sem revelar ao adversário tal posicionamento. O segundo tabuleiro representa a área do adversário com todas as posições encobertas (isto é, sem informações sobre o que se encontra lá). As jogadas se dão de forma alternada.

Quando da sua vez, um jogador "atirar" em uma posição do tabuleiro do adversário indicando as suas coordenadas (linha e coluna). O jogo deve então informar se o tiro acertou algo ou não e, se o jogador acertou algo e todas as partes de uma embarcação já foram atingidas após o último "tiro", então tal embarcação foi afundada e o jogo deve informar o tipo de embarcação que afundou. Ao longo do jogo, o segundo tabuleiro registra a título de controle, o resultado de cada "tiro" que deu no tabuleiro do adversário identificando as partes das embarcações já atingidas. Ganha o jogo quem afundar primeiro toda a frota do adversário.

2 Implementação

O jogo deve lê o posicionamento inicial das embarcações de um arquivo texto. O estado inicial do tabuleiro é armazenado em tal arquivo, onde, cada linha representa uma linha do tabuleiro. O tabuleiro terá 10 linhas e 10 colunas enumeradas de 1 a 10. **Dica:** Utilize os arquivos exemplo do arquivo exemplos_tp01.zip no SIGAA como base para sua implementação.

A quantidade e o formato de cada embarcação no tabuleiro são indicados na tabela abaixo:

Quantidade	Embarcação	Tamanho (Horizontal ou Vertical)	Representação
1	Porta-aviões	4 Casas	P
3	Submarino	3 Casas	S
3	Destroier	2 Casas	D
2	Bote	1 Casa	B

Table 1: Especificação das embarcações.

A Figura abaixo mostra um exemplo de arquivo de exemplo de distribuição da frota em um tabuleiro. Para representar a água é utilizado o símbolo ~ (til).

```
D~~~~~~~~S
D~~PPPP~~S
~~~~~~~~~S
~SSS~~DD~~
~~~~~S~~~~
~~~~~S~~~D
B~~~~S~~~D
~~~~~~~~~~
B~~~~~~~~~
~~~~~~~~~~
```

Figure 1: Exemplo de Mapa de um jogador.

2.1 Carregamento dos Tabuleiros

O programa dever ter uma função chamada carregarTabuleiros, que deve ler os arquivos dos jogadores e armazenar no tabuleiro de cada jogador (matriz). A função deve ter o seguinte protótipo:

```
1 int carregarTabuleiros(char arquivo1[], char arquivo2[], char
    tabuleiro1[NUM_LINHAS][NUM_COLUNAS], char tabuleiro2[NUM_LINHAS][NUM_COLUNAS]);
```

A função retorna 0 (zero) caso o carregamento dos arquivos sejam feito com sucesso e 1 (um) caso ocorra um erro de carregamento. Se houver um erro o programa deve parar a execução. Considere que NUM_LINHAS e NUM_COLUNAS são constantes com valores 10 e 11, respectivamente.

2.2 Mostrar tabuleiros

Para mostrar os tabuleiros na tela o programa deve chamar a função mostrarTabuleiros, que deve ler os tabuleiros (matrizes). Lembre que as embarcações do inimigo devem ficar ocultas no tabuleiro, apenas devem ser apresentadas as partes das embarcações já atingidas. No seu tabuleiro, mostre as representações seguindo as informações da Tabela 2 e as partes das embarcações já atingidas (se houverem). A função deve ter o seguinte protótipo:

```
1 void mostrarTabuleiros(char tabuleiro1[NUM_LINHAS][NUM_COLUNAS], char tabuleiro2
    [NUM_LINHAS][NUM_COLUNAS]);
```

A função dever imprimir os dois tabuleiros (cabe a você definir o padrão estético). Veja o exemplo da figura abaixo:

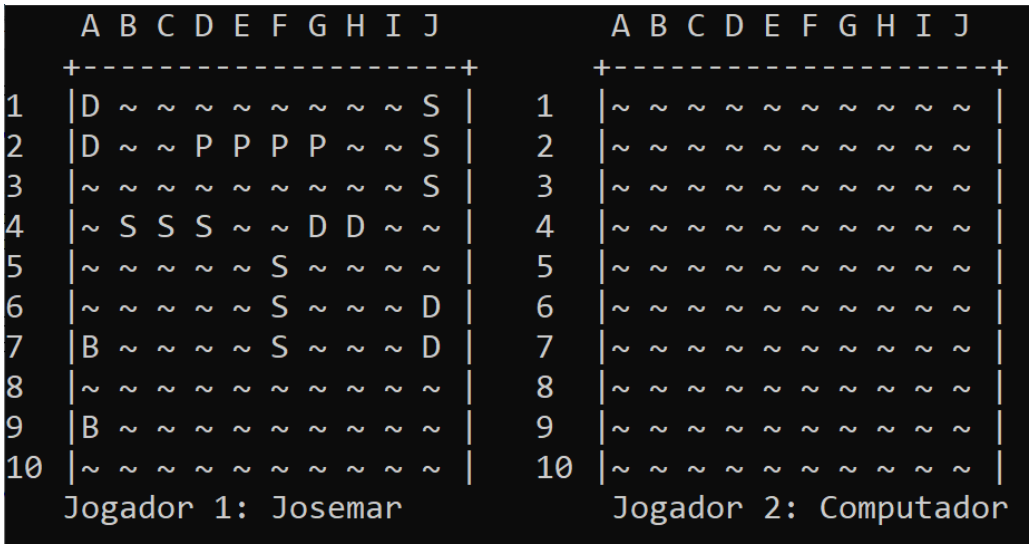

Figure 2: Exemplo de impressão da situação dos tabuleiros.

Quando uma parte da embarcação é atingida a representação da embarcação é substituída pelo símbolo * (asterisco). A figura abaixo mostra um tabuleiro com a representação de parte de uma embarcação atingida.

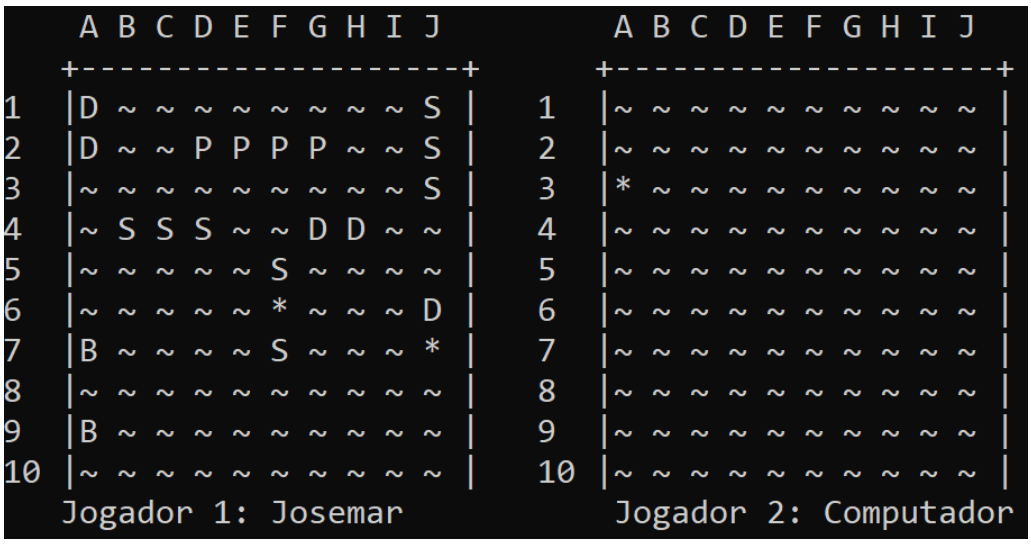

Figure 3: Exemplo de embarcações parcialmente destruídas.

Dica: Para limpar a tela em linguagem C, basta utilizar o seguinte comando:

```c
system("cls"); // inclua a biblioteca stdlib.h no inicio do seu programa
```

2.3 Atirar

O jogo deve solicitar ao jogador humano, na sua vez de jogar, que dispare 1 (um) "tiro", indicando as coordenadas do alvo através do número da linha e da letra da coluna que definem a posição. Ao jogador Computador, deve-se gerar aleatoriamente as coordenadas do tiro.

Para que o jogador tenha o controle dos "tiros" que acertam parte das embarcações, deverá marcar cada um deles na matriz do jogo (tabuleiro) com *. A função deve ter o seguinte protótipo:

```
1 int atirar (char tabuleiro[NUM_LINHAS][NUM_COLUNAS], int linha, int coluna);
```

A função deve retornar se o "tiro" acertou ou não parte de uma embarcação, retornando 0 (zero) quando acertar a água e 1 (um) quando acertar uma parte de uma embarcação. Quando acertar parte de uma embarcação o programa avisará que o "tiro" atingiu parte de uma embarcação. A função também retorna o valor 2 (dois) quando uma embarcação for totalmente destruída, se isso acontecer o programa informará qual embarcação foi afundada e as coordenadas dessa embarcação no tabuleiro será alterada, ou seja, o símbolo * (fogo) será substituído por x (destroços).

2.4 Execução

O programa (jogo) deve solicitar os seguintes parâmetros: (i) arquivo do jogador 1 (Humano), (ii) arquivo do jogador 2 (Computador) e (iii) Nome do jogador Humano. Em seguida, o jogo deve mostrar os tabuleiros (visão do Jogador Humano) e solicitar as coordenadas do tiro. Em seguida, o Jogador Computador gera aleatoriamente as coordenadas do tiro. O jogador que tiver todas as suas embarcações destruídas perde o jogo.

3 Funcionalidades Adicionais

Abaixo está uma lista de desafios de implementação de funcionalidades adicionais que valerão, no total, 5 pontos extras. Todas as funcionalidades implementadas devem estar no mesmo programa. **Dica:** reservem uma versão da solução básica antes de se aventurarem nos desafios.

- **Desafio 1:** O tabuleiro do Computador é gerado **aleatoriamente**, ou seja, o tabuleiro do Computador não é mais lido de arquivo. Você tem que garantir que o tabuleiro gerado é válido, ou seja, está de acordo com as especificações das embarcações (Tabela 1) (2 pontos);
- **Desafio 2:** Implemente um menu no início do jogo que possibilita escolher 3 níveis de dificuldade (referente ao Computador):
  - Fácil: o Computador gera aleatoriamente as coordenadas do tiro (podem se repetir);
  - Normal: o Computador gera aleatoriamente as coordenadas do tiro, porém somente escolhe coordenadas não repetidas (1 pontos);
  - Difícil (implementação de IA): o Computador gera aleatoriamente as coordenadas do tiro, porém ao acertar uma embarcação com mais de 1 casa de comprimento, na próxima jogada ele tenta atirar em uma casa vizinha na vertical ou horizontal (2 pontos);

4 Documentação

O texto da documentação deve ser breve, de forma que, eu (professor) possa entender o que foi feito no código sem ter que entender linha a linha do programa. Implementações modularizadas deverão mencionar quais lógicas são implementadas em cada módulo (função). A documentação deve conter, no mínimo, os seguintes itens:

- Nomes dos integrantes e a contribuição de cada um;
- Uma descrição dos algoritmos, das principais funções, e as decisões de implementação;
- Decisões de implementação (ou considerações) que porventura estejam omissos na especificação do trabalho;
- Testes, mostrando que o programa está funcionando de acordo com a especificação, seguidos da sua análise. *Print screens* mostrando o correto funcionamento do programa e exemplos de testes executados;

O arquivo da documentação deve estar em **.pdf**. Não serão aceitos outros formatos.

5 Critérios de avaliação

Os trabalhos serão avaliados de acordo com os critérios:

- Implementação Correta [0 a 8 pontos].
- Documentação [0 a 4 pontos].
- Interface (Usabilidade, estética, diversão, etc) [0 a 5 pontos];
- Tratamento de erros e Organização do código [0 a 3 pontos].
- Desafios terão nota 0, 25%, 50%, 75% ou 100% dos pontos extras de acordo com o funcionamento e adequação ao restante do código.
- **Cópias terão nota zero!**
- Trabalhos que não compilarem terão notas entre 0 e 5.
- Caso necessário, haverá uma entrevista com os integrantes do grupo.