RELATÓRIO TÉCNICO – Projeto Batalha Naval

1. Introdução

Este relatório descreve o projeto, a arquitetura e as decisões técnicas da implementação do jogo **Batalha Naval** em linguagem C.
O objetivo do trabalho é demonstrar domínio de:

- uso de **structs** para modelar entidades do jogo;
- uso de **ponteiros**, **alocação dinâmica** (malloc/realloc);
- organização modular do código (múltiplos `.c` e `.h`);
- validação de entradas, controle de memória e fluxo lógico completo de um jogo interativo.

O foco aqui é **explicar como o sistema foi estruturado**, e não ensinar o usuário a jogar.

---

2. Arquitetura do Projeto

O projeto foi dividido em módulos para facilitar manutenção, leitura e reuso. Cada módulo concentra uma responsabilidade única:

Módulos principais

- **board.c / board.h**
  Responsável pelo tabuleiro: criação, inicialização, impressão e validação de coordenadas.
- **ship.c / ship.h**
  Gerencia dados dos navios: tamanho, posição, estado (atingido/afundado).
- **fleet.c / fleet.h**
  Controla a frota de cada jogador: alocação, posicionamento e verificação de destruição

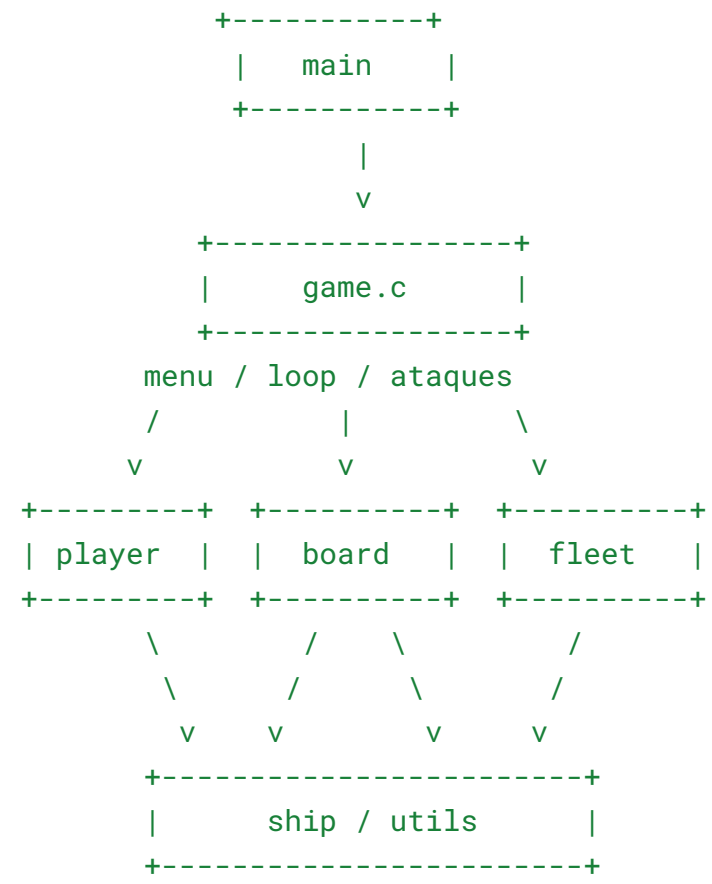
total.

- **player.c / player.h**
  Modela o jogador, contendo seu tabuleiro e sua frota.

- **game.c / game.h**
  Implementa o loop principal do jogo: menu, turnos, ataques, condições de vitória.

- **utils.c / utils.h**
  Funções auxiliares, como geração de números aleatórios ou leitura segura de inputs.

Fluxo de chamadas (diagrama ASCII)



A modularização garante que cada parte possa ser testada isoladamente e reduz dependências cruzadas.

---

3. Estruturas de Dados

Board (Tabuleiro)

Representa a grade do jogo.
Implementado como matriz dinâmica (`char**`), alocada com malloc:

- `'~'` → água
- `'N'` → navio
- `'X'` → acerto
- `'O'` → tiro na água

Por que dinâmica?

- Permite alterar dimensão no futuro (por exemplo, tabuleiros maiores).
- Exercita manipulação de ponteiros e acesso seguro.

Ship (Navio)

Estrutura contendo:

- tamanho
- quantidade de acertos
- coordenadas iniciais
- orientação (horizontal/vertical)

Usado para verificar se um navio foi completamente destruído.

Fleet (Frota)

É um vetor dinâmico de Ships.
Usamos `malloc` e, quando necessário, `realloc` para ajustar o número de navios.

Motivação:

- Permite adicionar novos tipos de navios sem reescrever código.
- Mantém o jogo escalável e organizado.

Player

Contém:

- seu `Board`
- sua `Fleet`
- quantidade total de navios restantes

Esse design deixa claro que cada jogador é um "pacote completo" de estruturas.

---

4. Fluxo de Execução

1. Menu Inicial

O usuário escolhe iniciar o jogo.

2. Configuração

O tabuleiro é criado e inicializado para os dois jogadores.
A frota é carregada e posicionada — manualmente ou de forma automática, dependendo da implementação.

3. Posicionamento

Antes do jogo começar:

- Verifica se a posição do ship é válida
- Verifica se cabe no tabuleiro
- Verifica se não sobrepõe outros navios

4. Loop de Turnos

Repetição principal:

1. O jogador seleciona coordenadas para atacar.
2. O sistema valida a entrada.
3. Atualiza o estado do tabuleiro inimigo.
4. Verifica se algum navio foi afundado.
5. Alterna o turno.

5. Detecção de Fim de Jogo

O jogo termina quando:

- todos os navios de um jogador são destruídos.

Essa validação é feita em `fleet.c`.

---

5. Decisões de Design

Decisões importantes que moldaram o sistema:

Modularização forte

Separar board, fleet, ship e game foi essencial para:

- evitar código gigante no main;
- permitir testes isolados;
- reduzir acoplamento.

Representação textual do tabuleiro

A escolha por `char**` dá controle total sobre alocação e facilita visualização.

Validação de coordenadas

A checagem é feita antes de qualquer operação, garantindo que o programa nunca acesse memória fora dos limites.

Orientação horizontal/vertical

Simplifica a lógica de colocação e deixa regras claras, reduzindo erros de sobreposição.

Uso de rand() em módulo separado

Mantém o código organizado e facilita futuras mudanças (ex: usar um algoritmo mais sofisticado de distribuição).

---

6. Gestão de Memória

A parte mais sensível do projeto.

Alocação

- Tabuleiros criados via `malloc` + loops que alocam cada linha.
- Frotas criadas com `malloc` e ajustadas com `realloc`.

Desalocação

- Cada linha do tabuleiro é liberada com `free`.
- A matriz em si é liberada depois.
- Cada ship e sua fleet são liberados ao final do jogo.
- O player chama free para suas estruturas internas.

Boas práticas adotadas

- Verificação de `NULL` após cada malloc.
- Nunca acessar índices fora da matriz.

- Nunca chamar `free` duas vezes no mesmo ponteiro.
- Uso de variáveis auxiliares para evitar ponteiros perdidos.

Testes contra vazamento

Utilização de cenários específicos com loops de posicionamento e ataques, buscando encontrar:

- acessos indevidos
- uso de ponteiros expirados
- estouro de índices

---

7. Testes e Validação

As principais estratégias de validação foram:

✔ Testes de posicionamento

- Navios fora dos limites
- Navios colidindo
- Orientação incorreta
- Possibilidade de adicionar navios grandes em espaços pequenos

✔ Testes de entrada

- Coordenadas inválidas
- Inputs não numéricos
- Repetição de ataques no mesmo ponto

✔ Testes de loop

- Jogo completo do início ao fim
- Troca de turnos
- Detecção de vitória funcionando corretamente

✔ Testes visuais do tabuleiro

- Impressão do estado
- Verificação de marcações (X e O)

---

8. Conclusão

O desenvolvimento do Batalha Naval permitiu aplicar conceitos centrais da disciplina:

- manipulação real de ponteiros
- modularização consistente
- structs bem definidas
- controle rigoroso de memória
- lógica estruturada para um sistema interativo

Desafios encontrados incluíram:

- alinhar o design modular com dependências entre módulos;
- garantir que o tabuleiro nunca fosse acessado fora dos limites;
- validar corretamente inputs do usuário;
- implementar free completo sem vazamentos.

Se fosse evoluir o projeto, seria interessante adicionar:

- tabuleiros maiores configuráveis
- IA mais inteligente
- interface gráfica
- sistema de logs para depuração

O projeto cumpre seu papel: demonstrar domínio técnico e entregar um jogo funcional, estável e bem estruturado.