RELATÓRIO TÉCNICO

Projeto: Implementação do Jogo Batalha Naval em C

Curso: Sistemas de Informação - CESAR School
Disciplina: Programação Imperativa e Funcional (PIF)
Período: 2025.2
Professor: João Victor Tinoco
Data de Entrega: 02/12/2025

---

1. EQUIPE DE DESENVOLVIMENTO

| Membro | Papel Principal | Contribuições Principais |

| **Danilo de Melo Duarte** | Líder de Projeto e Arquitetura | Estruturas de dados, lógica principal do jogo, validações | | **Pedro Pessoa Bastos** | Desenvolvedor de Interface | Sistema de I/O, sons simulados, menu interativo | | **João Gabriel de Souza Neri** | Especialista em Memória | Alocação dinâmica, tratamento de erros, otimização |

2. RESUMO DO PROJETO

Este projeto consiste na implementação completa do clássico jogo **Batalha Naval** utilizando exclusivamente a linguagem C e suas bibliotecas básicas. O objetivo principal foi consolidar os conceitos fundamentais de programação estruturada, com foco especial em:

- Estruturas de dados complexas (structs)
- Ponteiros e manipulação de endereços de memória
- Alocação dinâmica de memória (malloc, realloc, free)
- Modularização e organização de código
- Validação rigorosa de entradas

O jogo foi desenvolvido seguindo rigorosamente as especificações do professor, implementando todas as funcionalidades solicitadas e adicionando melhorias na experiência do usuário.

3. ESPECIFICAÇÕES TÉCNICAS

3.1 Ambiente de Desenvolvimento

- **Linguagem:** C (padrão C99)
- **Sistema Operacional:** Multiplataforma (Windows/Linux/macOS)
- **Compilador:** GCC (GNU Compiler Collection)
- **Bibliotecas Permitidas:**
  - `stdio.h` (entrada/saída padrão)
  - `stdlib.h` (alocação dinâmica)
  - `string.h` (manipulação de strings)
  - `time.h` (geração aleatória)
  - `ctype.h` (manipulação de caracteres)
  - `stdbool.h` (tipo booleano)

3.2 Requisitos Funcionais Implementados

1. ✅ Tabuleiro configurável entre 6x6 e 26x26 (padrão 10x10)
2. ✅ Frota mínima conforme especificação:
   - 1 Porta-aviões (5 células)
   - 1 Encouraçado (4 células)
   - 2 Cruzadores (3 células cada)
   - 2 Destroyers (2 células cada)
3. ✅ Posicionamento manual e automático de navios
4. ✅ Sistema de turnos alternados entre jogadores
5. ✅ Validação de coordenadas no formato "B5"
6. ✅ Detecção de acertos, erros e navios afundados
7. ✅ Estatísticas de jogo (tiros, acertos, precisão)
8. ✅ Interface CLI completa com menu interativo
9. ✅ Sons simulados com texto ASCII
10. ✅ Opção de saída durante o jogo

4. ARQUITETURA DO SISTEMA

4.1 Estrutura de Diretórios

projeto_batalha_naval/ ├── src/ │ ├── main.c # Ponto de entrada do programa │ ├── board.h/.c # Manipulação do tabuleiro │ ├── fleet.h/.c # Controle da frota de navios │ ├── game.h/.c # Regras e lógica do jogo │ ├── io.h/.c # Entrada/saída e interface │ └── rnd.h/.c # Geração de números aleatórios ├── obj/ # Objetos compilados ├── Makefile # Script de compilação └── README.md # Documentação do projeto

text

4.2 Estruturas de Dados Principais

```
// Estado de uma célula do tabuleiro
typedef enum {
    CELL_WATER,  // Água não atingida
    CELL_SHIP,   // Navio não atingido
    CELL_HIT,    // Acerto em navio
    CELL_MISS    // Tiro na água
} CellState;

// Célula individual do tabuleiro
typedef struct {
    CellState state;   // Estado atual
    int ship_id;       // ID do navio (-1 se não houver)
} Cell;

// Tabuleiro completo
typedef struct {
    int rows, cols;    // Dimensões
    Cell *cells;       // Matriz dinâmica de células
} Board;

// Navio individual
typedef struct {
    char name[20];     // Nome do navio
    int length;        // Tamanho em células
    int hits;          // Número de acertos
    int placed;        // Se já foi posicionado
} Ship;

// Frota completa de um jogador
typedef struct {
    Ship *ships;       // Array dinâmico de navios
    int count;         // Quantidade de navios
} Fleet;

// Jogador
typedef struct {
    Board *board;      // Tabuleiro com navios
    Board *shots;      // Tabuleiro de tiros
    Fleet *fleet;      // Frota do jogador
    char nickname[32]; // Nome do jogador
    int total_shots;   // Estatísticas
    int hits;          // Estatísticas
} Player;

// Estado completo do jogo
typedef struct {
    Player p1, p2;     // Dois jogadores
    int current_player;// Quem joga agora (0 ou 1)
    int game_over;     // 0 = em andamento, 1/2 = vencedor
    int board_size;    // Tamanho do tabuleiro
} Game;
5. MÓDULOS DO SISTEMA
5.1 board.c / board.h
Responsabilidade: Manipulação da matriz do tabuleiro.

Funções Principais:

board_create(): Aloca memória para um novo tabuleiro

board_destroy(): Libera memória do tabuleiro

board_place_ship(): Posiciona navio com validação
```

board_mark_shot(): Processa um tiro no tabuleiro

board_print(): Exibe o tabuleiro no console

Características:

Alocação dinâmica da matriz usando malloc()

Validação de limites e sobreposição

Sistema de coordenadas baseado em índices

5.2 fleet.c / fleet.h
Responsabilidade: Gerenciamento da frota de navios.

Funções Principais:

fleet_create_default(): Cria frota padrão com 6 navios

fleet_is_ship_sunk(): Verifica se navio foi afundado

fleet_mark_hit(): Registra acerto em navio

fleet_count_sunk_ships(): Conta navios afundados

Características:

Array dinâmico de navios

Controle de estado individual de cada navio

Nomes padronizados para experiência do usuário

5.3 game.c / game.h
Responsabilidade: Lógica principal do jogo.

Funções Principais:

game_create(): Inicializa estrutura do jogo

game_take_shot(): Processa um turno de tiro

game_is_over(): Verifica condição de vitória

game_print_status(): Exibe estatísticas do jogo

Características:

Controle de turnos alternados

Verificação de fim de jogo

Cálculo de estatísticas em tempo real

5.4 io.c / io.h
Responsabilidade: Interface com o usuário.

Funções Principais:

io_get_coordinates(): Lê e valida coordenadas

io_place_ships_manual(): Interface de posicionamento

io_show_game_over(): Tela final do jogo

Funções de som simulado com ASCII

Funções de som simulado com ASCII

Características:

Validação robusta de todas as entradas

Feedback visual claro e consistente

Sons simulados para imersão

Opção de saída a qualquer momento (tecla 'Q')

5.5 rnd.c / rnd.h
Responsabilidade: Geração de números aleatórios.

Funções Principais:

rnd_init(): Inicializa gerador com semente baseada no tempo

rnd_range(): Gera número aleatório em intervalo

rnd_orientation(): Escolhe orientação aleatória (H/V)

Características:

Uso de srand(time(NULL)) para aleatoriedade

Funções auxiliares para posicionamento automático

5.6 main.c
Responsabilidade: Ponto de entrada e fluxo principal.

Funções Principais:

main(): Loop principal do programa

start_new_game(): Inicia nova partida

show_instructions(): Exibe manual do jogo

game_settings(): Configurações personalizadas

Características:

Menu interativo com 4 opções

Fluxo completo do jogo

Tratamento de saída antecipada

6. ALGORITMOS IMPLEMENTADOS
6.1 Algoritmo de Posicionamento Aleatório
c
```
// Pseudocódigo do algoritmo
para cada navio na frota:
    tentativas = 0
    enquanto não posicionado E tentativas < 1000:
        linha = aleatório(0, rows-1)
        coluna = aleatório(0, cols-1)
        orientação = aleatório(HORIZONTAL, VERTICAL)

        se posição_valida(linha, coluna, tamanho, orientação):
            posicionar_navio(linha, coluna, orientação)
            marcado_como_posicionado = verdadeiro

        tentativas++
```

Complexidade: O(n × 1000) no pior caso, onde n é o número de navios.

6.2 Algoritmo de Detecção de Navio Afundado

```
c
// Pseudocódigo do algoritmo
função navio_afundado(id_navio):
    navio = obter_navio_por_id(id_navio)
    retornar navio.hits >= navio.length
```

Complexidade: O(1) para verificação individual.

6.3 Algoritmo de Conversão de Coordenadas

```
c
// Conversão "B5" → (linha=4, coluna=1)
coluna = toupper(coordenada[0]) - 'A'  // 'B' → 1
linha = atoi(coordenada+1) - 1         // "5" → 4
```

7. GERENCIAMENTO DE MEMÓRIA

7.1 Estratégias de Alocação

Alocação Dinâmica de Tabuleiros:

```
c
board->cells = (Cell*)malloc(rows * cols * sizeof(Cell));
```

Alocação Dinâmica de Frotas:

```
c
fleet->ships = (Ship*)malloc(fleet->count * sizeof(Ship));
```

Alocação Hierárquica:

```
text
Game → Player → Board → Cells
                Fleet → Ships
```

7.2 Estratégias de Liberação

Ordem Inversa à Alocação:

```
c
free(board->cells);
free(board);
```

Verificação de Ponteiros Nulos:

```
c
if (game->p1.board) board_destroy(game->p1.board);
```

Liberação Completa:

```
c
void game_destroy(Game *game) {
    if (game->p1.board) board_destroy(game->p1.board);
    if (game->p1.shots) board_destroy(game->p1.shots);
    if (game->p1.fleet) fleet_destroy(game->p1.fleet);
    // ... similar para p2
    free(game);
}
```

8. TESTES E VALIDAÇÕES

8.1 Testes Realizados

Testes de Entrada:

Coordenadas válidas ("A1", "J10")

Coordenadas inválidas ("A0", "K5", "B")

Caracteres especiais e espaços

Testes de Limites:

Tiros fora do tabuleiro

Posicionamento em bordas

Navios que ultrapassam limites

Testes de Lógica:

Multiplos tiros na mesma posição

Afundamento completo de navios

Detecção correta do vencedor

Testes de Memória:

Uso de Valgrind para detecção de leaks

Testes de estresse com múltiplas partidas

8.2 Validações Implementadas
c
```c
// Exemplo de validação de coordenadas
bool coordenada_valida(char *input, int max_rows, int max_cols) {
    // Verifica formato básico
    if (strlen(input) < 2) return false;

    // Verifica letra da coluna
    char col_char = toupper(input[0]);
    if (col_char < 'A' || col_char > 'Z') return false;

    // Verifica número da linha
    char *endptr;
    long linha = strtol(input + 1, &endptr, 10);
    if (endptr == input + 1) return false; // Não converteu número

    // Verifica limites
    int col = col_char - 'A';
    return (linha >= 1 && linha <= max_rows &&
            col >= 0 && col < max_cols);
}
```
9. DESAFIOS E SOLUÇÕES
9.1 Desafio: Posicionamento Aleatório Eficiente
Problema: Algoritmo ingênuo poderia ficar preso em loop infinito.

Solução: Limite de 1000 tentativas por navio e algoritmo que prioriza espaços livres.

9.2 Desafio: Interface Clara sem Gráficos
Problema: Como fornecer boa experiência apenas com texto.

Solução:

Sons simulados com ASCII art ([BOOM!], [SPLASH])

Legendas claras e consistentes

Cores via caracteres especiais (quando suportado)

9.3 Desafio: Gerenciamento de Memória Complexo
Problema: Múltiplos níveis de alocação dinâmica.

Solução:

Funções _create() e _destroy() simétricas

Verificação sistemática de ponteiros NULL

Uso de Valgrind para debug

10. CRITÉRIOS DE AVALIAÇÃO ATENDIDOS
10.1 Design e Organização (20 pontos)
 Modularização clara em 6 módulos

 Nomes de variáveis e funções descritivos

 Comentários adequados e documentação

 Separação de responsabilidades

10.2 Uso de Conceitos (35 pontos)
 Structs para todas as entidades principais

 Ponteiros em todas as passagens entre funções

 malloc(), realloc() e free() corretamente utilizados

 Enums para estados e constantes nomeadas

 Fluxos de controle bem estruturados

10.3 Fluxos e Regras (25 pontos)
 Loop principal de jogo funcional

 Sistema de turnos alternados

 Validação completa de todas as regras

 Detecção correta de vitória

 Estatísticas de jogo calculadas

10.4 Qualidade de Código (20 pontos)
 Compilação sem warnings

 Código legível e bem indentado

 Tratamento de erros robusto

 Sem vazamentos de memória (testado com Valgrind)

11. CONCLUSÕES E APRENDIZADOS
11.1 Conclusões Técnicas
O projeto demonstrou que é possível implementar um jogo completo e complexo utilizando apenas as ferramentas básicas da linguagem C.

11.2 Aprendizados da Equipe
Danilo: Aprofundamento no design de estruturas de dados complexas e na lógica de jogos.

Pedro: Desenvolvimento de interfaces de usuário eficazes em ambiente restrito (CLI).

João: Domínio do gerenciamento de memória em programas com múltiplos níveis de alocação.

11.3 Recomendações para Versões Futuras
Implementar IA para modo single player

Adicionar suporte a rede para multiplayer remoto

Criar sistema de salvamento/carregamento de partidas

Desenvolver interface gráfica simples usando ncurses

12. REFERÊNCIAS
Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language.

Material de aula da disciplina PIF 2025.2 - CESAR School.

```
Documentação oficial GCC: https://gcc.gnu.org/onlinedocs/

C Standard Library Reference.
```