UNIVERSIDADE ESTADUAL DO CEARÁ
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ERICK CARNEIRO DE OLIVEIRA ROCHA - 1836135
JONH WEBERT GALDÊNCIO DE LIMA - 1826167
JOSÉ RAFAEL LIMA E SILVA - 1841080

RELATÓRIO DO TRABALHO DE PROGRAMAÇÃO E ALGORITMOS

FORTALEZA
2025.2

SUMÁRIO

1 INTRODUÇÃO

O programa é uma implementação simples do jogo Batalha Naval em linguagem C. O jogo segue as regras tradicionais do Batalha Naval, na qual há dois jogadores, que incialmente alocam seus navios no tabuleiro, e partem para as rodadas de ataque, que consistem em atacar uma célula do tabuleiro do jogador adversário, para que destrua parte dos navios. O objetivo é simples, destruir todos os navios do jogador adversário, portanto, vence quem concluir o objetivo primeiro.

O programa incia com uma tela de Menu, imprimindo algumas opções para o usuário, dentre elas estão: novo jogo, carregar jogo, instruções e sair. Ao optar por um novo jogo, inicia-se a rodada de alocação dos navios de ambos os jogadores, após isso, as rodadas de ataque finalmente começam, a partir de agora, os jogadores alternam entre os turnos e só terminará caso um dos jogadores sair ou acabar de fato a partida.

Este ralatório abordará os principais aspectos do programa, desde a estrutura do projeto, até as funções que determinam a lógica pro de trás do jogo. Além disso, será evidenciado as decisões da equipe durante todo o processo de desenvolvimento do jogo de Batalha Naval.

2 ESTRUTURA DO PROJETO

O projeto é dividido em três pastas chamadas: “data”, “include” e “src”, na qual a pasta “data” possui os arquivos com os dados de salvamento e outros utilitários, a pasta “include” possui os headers (cabeçalhos) e a pasta “src” possui os arquivos (.c) com a implemenentação das funções. Optamos por dividir dessa forma para que ficasse mais organizado, e mais fácil de trabalhar, dividindo o código em pequenos blocos que se relacionam, possibilitando uma melhor legibilidade.

```
data/                    include/              src/
|-- partidas/            |-- jogo.h            |-- main.c
|   |-- historico.txt    |-- tabuleiro.h       |-- jogo.c
|   `-- save.bin         |-- modulos.h         |-- tabuleiro.c
|-- menu.txt             |-- navios.h          |-- navios.c
`-- instrucoes.txt       `-- salvar.h          `-- salvar.c
```

Figura 1: Árvore de arquivos

Na pasta data, colocamos o arquivos de menu.txt e instrucoes.txt, que contém artes em ASCII com seus respectivos nomes. Ademais, o arquivo save.bin contém os dados de uma partida salva, e o historico.txt contém os créditos, ou seja, dados coletados após o encerramento de uma partida, nesse caso, a quantidade de rodadas que a partida durou, o status dos jogadores (vencedor e perdedor), os acertos e erros de ambos.

Na pasta include, colocamos todos os arquivos (.h) que contém as declarações das funções utilizadas no programa. Vale ressaltar que o header com mais importância se chama: modulos.h, pois possui os structs que contém os dados necessários que devem ser salvos, e que facilitam o manuseio do código. Tais structs são: Celulas, Navios, Player e Jogo.

```
typedef struct{
        char impressao;
        int valor;

}Celulas;
```

```
typedef struct{
    int id, vida, tamanho_navio;
    int pos_incial[2];
    char representante, direcao;
}Navios;
```

```
typedef struct{
    Navios navios[total_navios];
    int acertos, erros;
    int navios_restantes;
}Player
```

```
typedef struct {
    int rodada, rodada_alocacao, vez, fim;
    int vencedor, perdedor, tam_historico;
    char **historico;
}Jogo;
```

Figura 2: Structs

O struct de Celulas está totalmente relacionado aos tabuleiros dos jogadores, então, por esse motivo ele recebe uma char de impressão, que será o caracter que será impresso no terminal na posição daquela célula, e o valor, que será referente ao id do navio que ocupa aquela célula do tabuleiro. Vale lembrar que o tabuleiro é estático com tamanho padrão 10x10

O struct de Navios contém o id, tamanho, posição inicial (x, y), o caracter representante, que serve de identificação visual e a direção na qual o navio está. As direções seguem a rosa dos ventos, ou seja, 'n' para norte, 's' para sul, 'l' para leste e 'o' para oeste. Na variável "total_navios" está armazenado o valor 4 globalmente, logo, existem quatro id's e quatro tamanhos diferentes, um para cada navio. Observe:

ID	Navio	Tamanho
1	Bote	2 células
2	Submarino	3 células
3	Navio-Tanque	4 células
4	Porta-Aviões	5 células

Figura 3: Tabela de Navios

O struct Player contém as informações dos jogadores, seus navios, acertos, erros e seus navios que restam (variável utilizada nas rodadas de ataque). Quando um jogador tem a variável "navios_restantes" igual a zero, significa que o jogo acabou e que ele perdeu. A critério de exemplo, veja a seguir um esboço do vetor de navios referente a um dos jogadores.

```
[   ][   ][   ][   ]
  |
   `--> id, tamanho, direcao.....
```

Figura 4: Navios

Cada elemento do vetor possui as informações referente ao navio de id escolhido pelo jogador na rodada de alocação. Isso permite a liberdade do jogador de

estabalecer a ordem que vai alocar seus navios.

O struct Jogo contém as informações do jogo, por exemplo, a variável “‘rodada_alocacao” que verifica se está na rodada de alocação dos navios, também possui a quantidade de rodadas da partida, a variável “vez” , que armazena o jogador da vez atual, o variável “tam_historico”, que aumenta dependendo do número de rodadas, e o ponteiro para ponteiro, que armazena o que aconteceu em cada rodada da partida, e as variáveis que armazenam os números dos jogadores, em seu status de vencedor ou perdedor.

Na pasta src, colocamos todos os arquivos (.c), que contém as implementações de todas as funções que foram definidas nos headers do programa. Cada arquivo (.c) está relacionado a um header, isso permitiu uma maior organização durante todo o processo de desenvolvimento. No arquivo main.c existe apenas a função que simula o Menu do jogo. No arquivo jogo.c existem as funções que lidam diretamente com o jogo, desde limpeza de terminal até a execução do fluxo. No arquivo tabuleiro.c existem as funções que mexem diretamente com uma célula dos tavuleiros de ambos os jogadores. No arquivo navios.c existem as funções que mexem diretamente com os navios de ambos os jogadores, diretamente ligados com a rodada de alocação e com os ataques durante a partida. No arquivo salvar.c, está definida as funções de salvar e carregar jogo.

Após falar da estrutura geral e dos structs que compõem o jogo, vamos partir para as funções que determinam o seu funcionamento.

3 FUNÇÕES

3.1 Inteiras

No total temos seis funções do tipo Int no código, quatro delas se referem a validações, e a outra desempenha um papel vital no jogo. A função “trocarVez”, como o nome já diz, alterna entre a vez de cada jogador, ela é eficiente desde a rodada de alocação dos navios até o fim do jogo. As funções de validação são as

seguintes:

• verificarAlocacao: faz uma verificação para saber se o navio já foi escolhido, por exemplo, se o jogador tentar posicionar duas vezes o mesmo navio no tabuleiro, essa função não irá permitir.

• verificarPosicao: faz uma verificação para saber se o navio que a pessoa quer posicionar ainda ocupa uma das posições do tabuleiro. Por exemplo, caso a pessoa posicione a navio na borda direita do tabuleiro e tente posicionar ele seguindo a direção leste, essa função não irá permitir. a mesma coisa ocorre caso o jogador tente sobrepor navios.

• verificarViabilidade: faz uma verificação para saber se a posição que o jogador posicionou o navio é viável, como os navios não podem se sobrepor, caso o jogador tente colocar em um ponto morto de tamanho de uma célula, tal que para qualquer das 4 direções a partir desse ponto morto estiver ocupando navios ou passar da borda, essa função não permitirá a alocação.

• verificarVida: faz uma verificação para saber se o navio atingido foi afundado. Nas rodadas de ataque essa função é valiosa, pois é irá validar a destruição de um navio, permitindo assim que o jogo se encerre após validar a destruição dos quatro navios inimigos.

• realizarPalpite: função que realiza o palpite do jogador, se retornar 1, significa que o jogador acertou a embarcação do jogador adiversário, caso contrário, retorna 0. Isso é útil para que o jogador que acertou faça palpites consecutivos até errar, ou destruir todos os navios do adversário.

3.2 Voids

As funções voids estão espalhadas pelo projeto, no total são dezesseis, normalmente modificam valores de cada variável ou matriz, por passagens por referências. São cruciais para o andamento do jogo. Inicialmente, vamos abordar sobre as fun-

ções presentes no Menu (arquivo main.c), elas são:

• novoJogo: essa função reseta todos os stats do struct e inicia a rodada de alocação, ele determina o passo incial para começar uma nova partida.

• salvarJogo: essa função abre o arquivo “save.bin”, e escreve nele todos os dados que precisamos salvar durante o jogo, estão inclusos todos os structs.

• carregarJogo: função que abre o arquivo “save.bin”, lê os dados salvos nele e atribui a cada um dos structs.

• execJogo: função que ocorre após puxar os dados salvos ou após alocar os navios (caso o usuário tenha optado por começar uma nova partida). Ela garante o fluxo do jogo, e só é encerrada caso o jogador force a saída do terminal, saia manualmente com a opção nas rodadas de ataque ou encerre de fato o jogo.

• instrucoes: essa função abre o arquivo “ìnstrucoes.txt”e mostra na tela os dados contidos nele.

• clear: A função clear limpa todo o terminal, importante para dar a sensação de atualização de frames, e também para que o terminal não fique completamente poluído.

• clearBuffer: essa função faz uma limpeza no buffer, caso o usuário entre com um dado inválido, essa função limpa automaticamente e evita erros de tipagem nas variáveis.

Vale ressaltar que essas funções citadas foram implementadas no arquivo jogo.c, pois, nós quisemos minimizar o conteúdo do arquivo main.c, a critério de organização do código.

As funções referentes aos navios são essenciais para definir o incio e andamento do jogo. Diante disso, vamos abordar sobre as implementações dessas funções no arquivo navios.c. Portanto, observe-as:

• alocarInicialmente: essa função é o ponto de partida para que os jogadores posi-

cionem seus navios nos tabuleiros, é nela que é pedido ao jogador o id, a posição incial (x, y), no qual x se refere às linhas e y às colunas, e a direção do navio, até que se esgote seu estoque de 4 navios. Ao passo que essa função ocorre, é utilizada também as funções de verificação, para que tudo siga as regras do jogo.

• prepararIds: função que prepara o vetor de navios do jogador, serve para a padronizar todo o vetor com ids iguais a -1, dessa forma, não há problema na primeira alocação, visto que nem um id de navio é igual a -1. Ela está diretamente ligada com a verificarNavio, visto que o id é aquilo que define se o navio já foi ou não alocado. Como não temos quantidades variadas pra cada navio, essa função é o ponto chave para o set inicial dos navios.

• alocarNavio: essa função faz definitavemente a alocação do navio no tabuleiro do jogador, como cada célula do tabuleiro é um struct com as variaáveis: int valor e char impressao, essa função atribui o caracter representante do navio à impressao e o id ao valor. Obedecendo é claro, as especificações de célula inicial e direção passadas pelo jogador.

As funções que se relacionam com o tabuleiro estão no arquivo tabuleiro.h, e são indispensáveis. Pois, são elas que dão vida ao jogo, não tem como jogar algo sem visualizar as modificações durante o processo. Diante disso, veja agora o que cada função presente no arquivo faz.

• montarTabuleiros: essa função monta os dois tabuleiros, ela atribui zero ao valor de todas a células dos dois tabuleiros, além disso, ela também atribui o caracter "~" à todas as impressões. Dessa forma, essa função é vital para o inicio do jogo, padronizando os sets de inicialização dos tabuleiros.

• imprimirTabuleiro: função que imprime na tela do terminal o tabuleiro, para cada caracter há uma cor, e isso vai ser abordado mais a frente neste relatório, afinal, as cores em ANSI dão um aspecto de diversão ao jogo.

• mudarRepTabuleiro: essa função muda as impressões do tabuleiro. Analise a se-

guinte situação: durante a alocação dos navios no tabuleiro, o jogador consegue visualizar suas alterações por conta da modificação na impressão de cada célula do tabuleiro. O problema real acontece que, se não houver uma modificação após acabar a rodada de alocação, o jogador adversário iria conseguir enxergar as células exatas onde o navio foi alocado. Diante desse cenário, é importante mudar essas impressões, por esse motivo, existe a função aqui apresentada.

• mudarRepNavio: função que muda a impressão do navio caso ele seja destruído. Nas rodadas de ataque, caso um navio seja bombardeado, a célula na qual foi atacada recebe o caracter “%” , porém, se o navio for destruído, o caracter irá mudar para “#” , facilitando a vizualização do jogador.

Todas as funções citadas até agora se relacionam e de certa forma, montam o set incial do jogo. Para que a função int chamada “relizarPalpite”, citada anteriormente possa determinar os acertos e erros de cada ataque. Agora, observe agora as funções implementadas no arquivo jogo.c, que são cruciais para determinar as informações presentes no arquivo “historico.txt”ao acabar a partida.

• criarCreditos: função que abre o arquivo e escreve todos os dados importantes, como por exemplo a quantidade de rodadas, o status de cada jogador, a quantidade de acertos e erros, e também o histórico de cada rodada.

• carregarCreditos: essa função abre esse arquivo e lê todas as informações presentes nele, é chamada no fim da partida.

4 IDEIAS, PROBLEMAS E DECISÕES

Durante o desenvolvimento do jogo, tivemos muitas ideais, e também muitos problemas. Vale exemplificar todos os impasses presentes durante a criação. Bom, começando pelas ideias, a mais bem sucedida foi a organização em três pastas diferentes, isso ajudou muito na leitura e organização do projeto. Contudo, trabalhar com pastas e headers era algo que não haviamos feito antes.

Diante da ideia de organizar o projeto em várias pastas, tivemos alguns problemas com compilação e linking. Basicamente, criando o projeto em dois sistemas operacionais diferentes (Windows e Linux), a compilação no windows era bem sucedida, porém no linux tinhamos erros de múltiplas declarações. Isso acontecia pelo fato de termos declarado algumas variáveis globais em um dos header. Para resolver, declaramos-as no formato “extern”dentro do header, e declaramos-as de novo em um arquivo (.c), isso fez com que pudessemos acessar essas variáveis em todos os arquivos (.c) com aquele header incluso. Dessa forma, o problema estava resolvido, e assim conseguimos compilar os arquivos.

```
gcc main.c tabuleiro.c navios.c salvar.c jogo.c -I .\include -o batalha-naval

clang main.c tabuleiro.c navios.c salvar.c jogo.c -I .\include -o batalha-naval
```

Figura 5: Flags de compilação

Outro problema que enfrentamos durante o processo foi a dependência de headers, basicamente estavamos fazendo uma troca mútua entre os headers, isso criava um loop e não deixava criarmos os executáveis. Foi ai que decidimos pesquisar e entender como tirar essas dependências, chamadas de cíclicas. Depois de um tempo, retiramos criando o modulos.h, que armazena as declarções dos structs e variáveis globais usadas durante todo o código.

Outra ideia que tivemos foi colocar cores no terminal, e dar um aspecto de vida ao jogo. Porém, isso gerou alguns problemas. Apesar da ideia de colorir o jogo ser boa, não sabiamos como fazer isso, por esse motivo, entramos no stackoverflow e encontramos códigos capazes de modificar as cores de um printf no terminal. Os códigos são:

```
#define Verde     "\033[32m"
#define Amarela   "\033[33m"
#define Azul      "\033[34m"
```

Figura 6: Cores ANSI

Uma terceira ideia surgiu no fim do projeto, utilizar alocação dinâmica no histórico do jogo, pois não sabemos em quantas rodadas a partida irá acabar. Logo, utilizar alocação dinâmica para mostrar o histórico de ataques parecia ser bem interessante. Basicamente, a cada rodada o jogador pode acertar ou errar, fazendo o histórico seguir a seguinte configuração. O jogador que acertou deve receber um "Acertou" , o oponente deve receber "Bloqueado" e rodada deve aumentar. Caso jogador erre, ele deve receber um "Errou"'e mudar a vez. O problema foi salvar os dados do histórico em um arquivo txt, tivemos que dar um jeito de última hora, e foi sim possível salvar no "save.bin".

5 CONCLUSÃO

Inicialmente, criamos um repositório no Github para guardar nossas alterações durante o processo de criação do jogo. Nossos encontros foram durante toda a semana, para que pudessemos estabelecer as ideias e de fato colocar em prática. O código foi feito na maioria das vezes utiizando uma extensão chamada "Live Share"do Vscode.

Fizemos tudo em conjunto, para que a equipe ficasse unida em cada decisão, votações estavam sempre sendo feitas para qualquer ideia de criação. Dessa forma, consolidamos um belo trabalho em equipe.

Gradualmente, fomos implementando as funções e lógica do jogo, detalhando os processos e otimizando o código. Utilizamos tanto o conhecimento adquirido em sala de aula, quanto conhecimento externo com pesquisas em sites de cursos ou de documentações. Além disso, o auxilio dos monitores foi de grande ajuda, tanto em ideias quanto em conceitos da linguagem.

Apesar dos desafios durante a implementação, persistimos e conseguimos criar o batalha naval totalmente jogável. Com isso, melhoramos nossas habilidades individuais e em equipe, resultando em um projeto bem-sucedido.

REFERÊNCIAS

Headers (cabeçalhos) - O que são, para que servem, como criar e usar seus arquivos .h. Disponível em: <https://www.cprogressivo.net/2013/09/Header-cabecalho-o-que-sao-para-que-servem-como-criar-e-usar-seus-arquivos-.h.html>. Acesso em: 19 nov. 2025.

Text to ASCII Art Generator (TAAG). Disponível em: <https://patorjk.com/software/taag> Acesso em: 19 nov. 2025.

Gerenciamento de Dependências de Arquivos de Cabeçalho em C | LabEx. Disponível em: <https://labex.io/pt/tutorials/c-how-to-manage-header-file-dependencies-419184>. Acesso em: 21 nov. 2025.

Como destacar texto (mudar cor) em C ANSI. Disponível em: <https://pt.stackoverflow.com/questions/157898/como-destacar-texto-mudar-cor-em-c-ansi>. Acesso em: 1 dez. 2025.

MACEJKOVIC, P. **Como corrigir o erro com várias definições de uma função em C ++**. Disponível em: <https://pt.cyberaxe.org/article/how-to-fix-error-with-multiple-definitions-of-a-function-in-c>. Acesso em: 7 dez. 2025.

Qual a melhor maneira de se limpar o buffer do teclado. Disponível em: <https://pt.stackoverflow.com/questions/321285/qual-a-melhor-maneira-de-se-limpar-o-buffer-do-teclado>. Acesso em: 7 dez. 2025.

ANTIQUEIRA, L.; LIANG, N.; BÁSICAS. **SCC-814-Projetos de Algoritmos Revisão da Linguagem C**. [s.l: s.n.]. Disponível em: <http://wiki.icmc.usp.br/images/2/29/SCC0814-Revisão_C.pdf>. Acesso em: 14 dez. 2025.

FERNANDA, A.; DE, V. **Fundamentos da Programação de Computadores**.