

CTeSP Tecnologias e Programação de Sistemas de Informação



Programação 2025/26

Projeto Final – Jogo da Batalha Naval



Gustavo Reis Dias, 25858

ÍNDICE GERAL

ÍNDICE DE FIGURAS	ii
ÍNDICE DE TABELAS	iii
CAPÍTULO I	4
1.1. Objetivos	4
1.2. Estrutura do Programa	5
1.3. Estrutura de Dados.....	6
CAPÍTULO II	8
2.1. Exemplos de Execução do Jogo.....	Erro! Marcador não definido.
2.2. Manual do Utilizador.....	10
Conclusões e Melhorias Futuras	13
Declaração de uso de ferramentas de geração de código e/ou modelos de linguagem	14
REFERÊNCIAS BIBLIOGRÁFICAS	15

ÍNDICE DE FIGURAS

Figura 1 - Introdução de Coordenadas.....	Erro! Marcador não definido.
Figura 2 - Menu inicial do jogo.....	12

ÍNDICE DE TABELAS

Tabela 1 - Listagem de Testes Realizados	8
--	---

CAPÍTULO I

1.1. Objetivos

Este capítulo descreve o propósito e as metas estabelecidas para o desenvolvimento do projeto "Batalha Naval". O objetivo principal é a criação de um programa interativo que utilize a linguagem C para simular a mecânica do jogo tradicional, demonstrando a aplicação prática de conceitos fundamentais da programação estruturada.

Com este trabalho, pretende-se demonstrar competências específicas na estruturação lógica de programas complexos, na gestão eficiente do estado do jogo (recorrendo a matrizes e estruturas de dados personalizadas) e na implementação de mecanismos rigorosos de validação de entradas do utilizador, garantindo que o software reage de forma previsível mesmo perante dados incorretos.

Requisitos Funcionais Implementados (Obrigatórios)

A aplicação cumpre integralmente os requisitos base definidos para o projeto:

Colocação de Navios: Algoritmo para o posicionamento automático de uma frota de 10 navios de diferentes dimensões no tabuleiro.

Mecânica de Disparos: Sistema que permite ao utilizador atacar coordenadas específicas.

Deteção de Eventos: Identificação e resposta imediata a "Água", "Acertos" e "Afundamentos" de embarcações.

Condição de Vitória: Verificação contínua do estado da frota para declarar o fim da partida quando todos os alvos forem destruídos.

Gravação de Pontuações (Persistência): Implementação de um sistema de ficheiros para registar o histórico de melhores resultados.

Sistema de Pontuação Dinâmico: Cálculo de pontos baseado no rácio de eficácia entre disparos efetuados e navios destruídos.

Instruções Integradas: Menu dedicado a explicar as regras ao utilizador diretamente na consola.

Requisitos Não Funcionais

O projeto priorizou padrões de qualidade de software, nomeadamente:

Robustez: Capacidade de processar e recuperar de erros de introdução (como o uso de letras onde se esperam números), recorrendo a técnicas de limpeza de *buffer*.

Legibilidade: Código organizado com indentação consistente, comentários explicativos e nomes de variáveis semânticos.

Modularidade: Organização do código em módulos (.c e .h), separando a interface da lógica de dados para facilitar futuras expansões.

1.2. Estrutura do Programa

A organização do programa foi concebida sob uma arquitetura modular, dividindo o código em ficheiros especializados que separam a interface, a lógica de negócio e a persistência de dados. Esta divisão facilita significativamente a manutenção, permitindo isolar erros e compreender a hierarquia de chamadas de forma intuitiva.

Organização por Ficheiros e Módulos

O código está distribuído em três unidades lógicas:

main.c: O ponto de entrada da aplicação. Gere o menu de estados e a interação inicial com o utilizador.

Jogo.c / Jogo.h: O motor central do jogo. Contém as definições de constantes (dimensões e símbolos) e todas as funções que manipulam a mecânica de combate.

scores.c / scores.h: Módulo dedicado exclusivamente à gestão de ficheiros, isolando a lógica de leitura e escrita do resto do programa.

Fluxo Principal do Jogo

O fluxo segue uma sequência lógica rigorosa, desde o arranque até ao encerramento:

Inicialização: O programa configura a semente aleatória (srand) e invoca inicializarTabuleiro, que preenche a matriz 10 X 10 com o símbolo de mar (~).

Colocação da Frota: A função colocarFrotaAleatoria percorre um array de tamanhos pré-definidos (de Porta-aviões a Submarinos). Para cada navio, a "IA" gera coordenadas e orientações aleatórias, utilizando verificarPosicao para garantir que as regras de limites e não sobreposição são respeitadas.

Ciclo de Partida (jogarPartida): O utilizador entra num ciclo interativo onde é solicitado a introduzir coordenadas. O tabuleiro é impresso em cada iteração através de mostrarTabuleiro.

Atualização e Verificação: Cada tiro é validado por validarJogada. Se for um acerto, o símbolo na matriz muda para X e o programa percorre a estrutura da frota para atualizar os danos. Se o navio esgotar as suas células intactas, é declarado afundado.

Condição de Fim: O ciclo termina quando naviosAfundados == 10 (vitória) ou numTiros == 50 (derrota). No desfecho, a pontuação é calculada e enviada para o módulo de scores.

Responsabilidade das Funções Principais

A separação de responsabilidades é garantida por funções com propósitos únicos:

verificarPosicao: Essencial para a integridade do mapa, valida se um navio "cabe" no espaço gerado antes da sua escrita física na matriz.

validarJogada: Atua como um filtro de segurança, impedindo acessos fora da memória do array e filtrando tentativas de disparar em coordenadas já reveladas.

jogarPartida: Coordena o estado global da sessão, gerindo os contadores de tiros e navios, além de controlar a transição entre o pedido de input e o feedback visual.

mostrarTabuleiro: Separa a representação de dados da visualização, permitindo ocultar os navios não atingidos através de um parâmetro de controle.

Lógica de Posicionamento do Computador

A lógica adotada para os disparos do computador não existe no ataque (pois o jogo é a solo), mas é crítica na fase de defesa. A colocação da frota utiliza uma **estratégia de tentativa e erro (brute-force aleatório)**: o computador gera coordenadas `rand() % 10` e apenas confirma a posição se a função de validação retornar sucesso. Esta lógica assegura que cada partida apresenta um desafio único e imprevisível para o jogador.

1.3. Estrutura de Dados

A representação interna da informação foi estruturada para garantir eficiência no processamento e clareza na lógica de jogo. Optou-se por uma arquitetura que utiliza uma matriz para o espaço visual e um array de estruturas para o estado lógico.

Representação do Tabuleiro (Matriz Bidimensional)

O tabuleiro é representado por um array bidimensional de caracteres (`char tabuleiro[TAM_TABULEIRO][TAM_TABULEIRO]`), dimensionado com a constante `TAM_TABULEIRO` definida como 10. Esta estrutura matricial é a escolha ideal para representar coordenadas cartesianas (X, Y). Cada célula armazena um valor que codifica o seu estado atual:

~ (**SIMBOLO_MAR**): Célula virgem, onde o conteúdo é desconhecido para o jogador.

* (**SIMBOLO_AGUA**): Indica um tiro falhado, marcando o local onde não existe qualquer embarcação.

X (**SIMBOLO_ACERTO**): Assinala uma coordenada onde um navio foi atingido com sucesso.

N (**SIMBOLO_NAVIO**): Identifica internamente a presença de uma parte de um navio.

Gestão de Navios e Detecção de Afundamento

Embora a matriz identifique as posições ocupadas, foi implementada uma estrutura adicional (`struct Navio`) para gerir a lógica de vida de cada embarcação. Cada um dos 10 navios da frota possui os seguintes campos:

Tamanho: Comprimento do navio (de 1 a 4 células).

Acertos: Contador de partes atingidas.

Linha / Coluna: Coordenadas de origem no tabuleiro.

Horizontal: Flag booleana que define o eixo de expansão.

Porquê usar uma estrutura adicional?

Se usássemos apenas a matriz, determinar se um navio foi afundado exigiria percorrer todo o tabuleiro em busca de células N adjacentes a cada disparo. Com a `struct Navio`, basta incrementar o campo `acertos` da embarcação correspondente à coordenada atingida. Quando `acertos == tamanho`, o sistema sabe instantaneamente que o navio foi afundado, permitindo atualizar o contador global de forma eficiente.

Visão do Adversário e Segurança do Jogo

A implementação da "visão do adversário" é feita através de uma impressão filtrada. Em vez de manter dois tabuleiros separados (um real e um de tiros), a função `mostrarTabuleiro` recebe um parâmetro de controlo. Se o jogo estiver em curso, a função percorre a matriz e, ao encontrar o símbolo N, imprime ~. Isto garante que o jogador nunca vê os navios ocultos, mas o sistema pode continuar a processar as colisões na mesma estrutura.

Prevenção de Jogadas Inválidas

A integridade da partida é assegurada pela função `validarJogada`. Esta utiliza a matriz para impedir disparos repetidos: antes de processar qualquer tiro, o sistema verifica se o carácter na posição é * ou X. Caso seja, a jogada é rejeitada. Além disso, as dimensões dos arrays são rigorosamente respeitadas através do uso de constantes, garantindo que o posicionamento aleatório e os tiros do utilizador nunca tentam aceder a endereços de memória fora dos limites definidos para o tabuleiro 10x10.

CAPÍTULO II

2.1. Exemplos de Execução do Jogo

Este capítulo apresenta o comportamento do software em execução, demonstrando que o sistema cumpre as regras de negócio e validações previstas. Abaixo, apresentam-se as transcrições das fases principais do jogo.

Menu Inicial e Inicialização: Ao iniciar a aplicação, o utilizador é confrontado com o menu principal. A escolha da opção '1' desencadeia a inicialização do tabuleiro e a colocação interna (oculta) da frota

```
=====
                        BATALHA NAVAL
=====
1 - Novo Jogo
2 - Consultar Pontuacoes
3 - Instrucoes
0 - Sair
Escolha uma opcao: 1

--- BATALHA NAVAL ---
Afunde os navios escondidos do computador!

  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Tiros: 0/50 | Afundados: 0/10
Coordenadas (Linha Coluna):
```

B. Sequência de Disparos e Detecção de Impacto: O sistema valida cada tiro. Caso o utilizador acerte num navio de tamanho 1 (submarino), o programa atualiza a matriz visual e decrementa a frota restante.

```
Coordenadas (Linha Coluna): 2 3

--> ACERTOU!
*** NAVIO AFUNDADO! (Tamanho 1) ***

  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2 ~ ~ ~ X ~ ~ ~ ~ ~ ~
3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Tiros: 1/50 | Afundados: 1/10
```

C. Validação de Erros e Correção de Entrada: O programa está blindado contra coordenadas fora dos limites (ex: 12 5) e disparos em células já exploradas. O sistema emite um erro e solicita uma nova entrada válida sem interromper o fluxo.

```
Coordenadas (Linha Coluna): 12 5
Erro: Coordenadas fora do tabuleiro!
Coordenadas (Linha Coluna): 2 3
Erro: Já disparou aqui!
Coordenadas (Linha Coluna): 4 4

--> AGUA!

  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
2 ~ ~ ~ X ~ ~ ~ ~ ~ ~
3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
4 ~ ~ ~ ~ * ~ ~ ~ ~ ~
5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

D. Encerramento e Vitória: Quando o décimo navio é afundado, o jogo termina com uma mensagem de sucesso e solicita o nome para o registo de pontuação.

```
...
*** NAVIO AFUNDADO! (Tamanho 2) ***

PARABENS! Venceu.

--- FIM DE JOGO ---
A sua pontuacao final: 1250 pontos
Introduza o seu nome (uma palavra): Jose
Pontuacao gravada com sucesso.
```

Tabela 1 - Listagem de Testes Realizados

ID do Teste	Descrição do Teste	Condições de Entrada	Resultado Esperado	Resultado Obtido
T01	Inicialização do Mapa	Jogo iniciado	Matriz 10x10 preenchida com '~'	Conforme esperado
T02	Coordenada Inválida	Linha 12, Coluna 5	Erro e novo pedido	Conforme esperado
T03	Entrada Alfabética	'abc'	Captura pelo buffer e novo pedido	Conforme esperado
T04	Tiro Repetido	Coordenada já usada	Erro "Já disparou aqui"	Conforme esperado
T05	Deteção de Vitória	10 Navios Afundados	Mensagem de parabéns e gravação	Conforme esperado

2.2. Manual do Utilizador

Este capítulo serve como guia prático para o utilizador final, descrevendo detalhadamente as regras de interação e a interpretação visual do jogo.

Formato de Introdução de Coordenadas

Para realizar um disparo, o utilizador deve introduzir dois números inteiros separados por um espaço, correspondentes à Linha e à Coluna, respetivamente. O sistema utiliza um índice de 0 a 9 para ambas as dimensões.

Exemplo Correto: 2 3 (Dispara na linha 2, coluna 3).

Exemplos de Erros Típicos:

12 5: Coordenada fora dos limites do tabuleiro.

A 1: Uso de caracteres alfabéticos em vez de numéricos.

5,2: Uso de vírgula em vez de espaço.

Resposta do Programa a Erros e Entradas Inválidas

O software foi desenhado para ser resiliente a falhas de digitação:

Valores Inválidos: Caso o utilizador insira coordenadas fora do intervalo 0, 9, o programa exhibe a mensagem *"Erro: Coordenadas fora do tabuleiro!"* e solicita novos dados.

Disparos Repetidos: Se o utilizador tentar disparar numa célula que já contém * ou X, o programa avisa *"Erro: Já disparou aqui!"*, não gastando munição.

Entradas Non Numéricas: Se forem introduzidas letras ou símbolos, o programa utiliza um mecanismo de limpeza de *buffer* (através da variável lixo) para descartar o texto inválido, impedindo o bloqueio do sistema e permitindo que o utilizador tente novamente.

Representação Visual e Legenda de Símbolos

O tabuleiro é apresentado no ecrã como uma grelha numerada de 0 a 9 nos eixos X e Y para facilitar a orientação. O significado de cada símbolo é o seguinte:

~: **Mar/Desconhecido.** Representa uma coordenada onde ainda não foi efetuado qualquer disparo.

*: **Água.** Indica um tiro falhado (não atingiu nenhum navio).

X: **Acerto.** Indica que uma parte de um navio foi atingida.

N: **Navio.** Representa a posição de uma embarcação. Por questões de jogabilidade, este símbolo permanece oculto (substituído por ~) até que seja atingido ou até que o jogo termine.

Distinção de Tabuleiros

Nesta versão do jogo (Utilizador vs Computador), o ecrã exhibe o Registo de Tiros do jogador contra a frota oculta do adversário.

Tabuleiro do Utilizador: Não é exibido graficamente durante o jogo, pois o computador não efetua disparos nesta versão; o foco é inteiramente na ofensiva do utilizador.

Tabuleiro do "Adversário": É a grelha principal onde o jogador tenta localizar as embarcações. No final da partida (em caso de derrota por falta de munições), o programa revela a posição final de todos os navios (N) que não foram afundados, permitindo ao utilizador verificar a estratégia do computador.

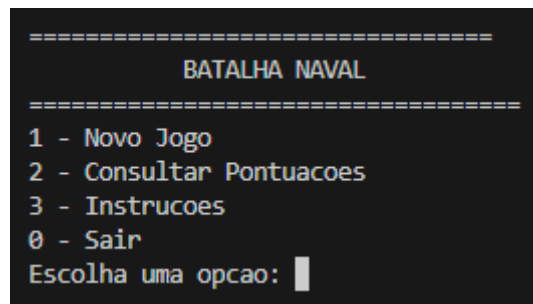


Figura 1 - Menu inicial do jogo

Resolução de Problemas e Guia de Execução

Para garantir que o programa é executado corretamente, o utilizador deve seguir o guia abaixo para compilação e resolução de erros comuns.

Guia de Compilação e Execução: Para compilar o programa de forma correta, é necessário incluir todos os módulos .c. Utilize o seguinte comando no terminal (ex: VS Code Terminal): `gcc main.c Jogo.c scores.c -o jogo.exe`

Para executar, digite: `.\jogo.exe` (Windows) ou `./jogo.exe` (Linux/Mac)

Erros de Compilação Comuns:

"Undefined reference to...": Ocorre se esquecer de incluir algum ficheiro no comando gcc (por exemplo, omitir o scores.c). Certifique-se de listar os três ficheiros referidos acima.

Falta de permissões: Se o compilador falhar ao criar o jogo.exe, verifique se o ficheiro não está aberto noutra janela ou se tem permissões de escrita na pasta.

Problemas em Tempo de Execução

Execução no diretório errado: O jogo deve ser executado na pasta onde se encontram os ficheiros de código. Se for executado noutra localização, o programa poderá não conseguir criar ou ler o ficheiro scores.txt corretamente.

Ficheiro de Scores Não Encontrado: O programa cria automaticamente o ficheiro scores.txt se ele não existir. No entanto, se o programa não mostrar a tabela de pontuações, verifique se o ficheiro não está bloqueado como "Apenas Leitura" pelo sistema operativo.

Comportamento Inesperado (Inputs): Caso o menu principal pareça "saltar" ou repetir-se infinitamente, isso deve-se a uma entrada de texto inválida no scanf. O programa tenta recuperar automaticamente através da variável lixo. Se isto acontecer, basta pressionar Enter e introduzir um número válido.

CONCLUSÕES E MELHORIAS FUTURAS

O desenvolvimento deste projeto permitiu alcançar com sucesso todos os objetivos propostos, resultando numa aplicação funcional, modular e robusta. A síntese do trabalho reflete a capacidade de transformar regras de jogo lógicas em algoritmos eficientes através da linguagem C.

Cumprimento de Objetivos e Reflexão Técnica

Os requisitos funcionais, como a geração aleatória da frota, a detecção de acertos e a persistência de dados em ficheiro, foram integralmente cumpridos. A escolha de uma arquitetura modular revelou-se fundamental: ao separar a lógica de combate (Jogo.c) da gestão de pontuações (scores.c), foi possível desenvolver e testar cada componente isoladamente, o que reduziu drasticamente o tempo de depuração e aumentou a clareza do código.

Aspetos Desafiantes e Soluções Adotadas

O desenvolvimento apresentou desafios técnicos significativos que exigiram soluções criativas:

Gestão Simultânea de Representações: O maior desafio residiu em sincronizar a matriz visual (char) com a estrutura lógica (struct Navio). Manter a "visão filtrada" do adversário sem perder a capacidade de rastrear a saúde individual de cada navio exigiu uma lógica de pesquisa linear eficiente a cada disparo.

Colocação Aleatória com Restrições: Implementar o algoritmo que posiciona os navios respeitando limites e sobreposições foi um processo complexo. Foi necessário criar uma lógica de "tentativa e erro" onde a função verificarPosicao atua como um guardião antes de qualquer alteração na memória da matriz.

Validação de Coordenadas e Robustez: Garantir que o programa não bloqueasse perante caracteres alfabéticos (erros de tipo no scanf) foi essencial. A implementação da variável de limpeza de *buffer* (lixo) foi a solução encontrada para manter o programa em execução mesmo sob utilização incorreta.

Melhorias Futuras

Embora o programa esteja totalmente funcional, existem áreas para evolução futura:

Inteligência Artificial de Ataque: Implementar um "modo CPU" onde o computador também efetua disparos contra o jogador, utilizando algoritmos de procura em torno de acertos confirmados.

Refinamento da Colocação: Adicionar uma regra de proximidade que impede que dois navios sejam gerados em células adjacentes, tornando o tabuleiro mais equilibrado.

Interface Gráfica: Evoluir do modo texto para uma interface baseada em caracteres coloridos ou bibliotecas gráficas simples para melhorar a imersão.

DECLARAÇÃO DE USO DE FERRAMENTAS DE GERAÇÃO DE CÓDIGO E/OU MODELOS DE LINGUAGEM

Este capítulo visa garantir a transparência no desenvolvimento do projeto, identificando o apoio de ferramentas de inteligência artificial durante a sua concepção técnica.

Identificação da Ferramenta

Ferramenta: Gemini (Google).

Modelo/Ambiente: Gemini 2.5 Flash.

Período de Utilização: janeiro de 2026.

Finalidade: Apoio na arquitetura do código, depuração de erros de input e lógica algorítmica.

Lista de Prompts Técnicos (Cronológica)

"Cria um código de Batalha Naval em C para um utilizador contra computador usando structs para os navios e de acordo com o enunciado."

"implementa a função verificarPosicao para garantir que os navios não se sobrepõem"

"Implementa uma lógica de detecção de impacto que identifique se a coordenada atingida pertence a um navio horizontal ou vertical."

"Cria uma solução para limpar o buffer do teclado para evitar loops infinitos com o scanf."

"Explica matematicamente como o operador módulo (%) limita o rand() às dimensões do tabuleiro."

"Ajuda-me a implementar a lógica de detecção de 'afundado' incrementando o campo 'acertos' na struct."

"Gera a estrutura base para um relatório técnico que descreva este código final."

Integração e Modificações Realizadas

Os *outputs* gerados foram fundamentais para a estabilidade técnica do projeto, tendo sido integrados e adaptados da seguinte forma:

Arquitetura e Lógica: A ferramenta propôs a separação entre a matriz visual e o array de estruturas Navio. O estudante validou esta sugestão e procedeu à refatorização integral de todas as funções para português (ex: `check_position` para `verificarPosicao`).

Tratamento de Erros: A solução para o erro de tipo no scanf foi sugerida pela ferramenta, mas a sua implementação foi testada e ajustada pelo estudante para garantir que o utilizador recebesse mensagens de erro claras antes da limpeza do *buffer*.

Modularização: A IA ajudou a definir os protótipos nos ficheiros .h. O estudante foi responsável por organizar fisicamente os ficheiros e resolver dependências de compilação.

Relatório: A ferramenta foi utilizada para gerar os esboços iniciais das explicações técnicas (especialmente a matemática do módulo e a lógica de deteção). O estudante procedeu à reescrita e expansão manual de todos os capítulos para refletir as decisões finais de design e os resultados dos testes de execução.

Responsabilidade

O autor assume responsabilidade total pelo trabalho final entregue. Foram realizadas verificações manuais de segurança, auditoria de lógica de ponteiros e testes de consistência funcional no terminal para garantir a originalidade e a conformidade integral com o enunciado.

REFERÊNCIAS BIBLIOGRÁFICAS

Slides de Programação 2025/26 - ESTG Portalegre.