

**UNIVERSIDADE DO SUL DE SANTA CATARINA**

**Erick Vieira**

**Patrick Lohn**

**Ruan Oliveira**

**Rafael Sonoki Swogo Zamora**

**A3 Estrutura de Dados e Análise de Algoritmos:**

**Documentação DO PROJETO - BATALHA NAVAL**

Orientador: Jorge Werner

Palhoça, 27 de abril de 2025

Sumário

[Documento do Projeto - Batalha Naval 3](#_Toc196301101)

[1. Nome e Tema 3](#_Toc196301102)

[2. Integrantes e Papéis 3](#_Toc196301103)

[3. Descrição do Funcionamento 3](#_Toc196301104)

[4. Uso de Estruturas de Dados 4](#_Toc196301105)

[5. Análise de Complexidade (Big O) 4](#_Toc196301106)

[Por que analisamos deque.rotate()? 5](#_Toc196301107)

# Documento do Projeto - Batalha Naval

## 1. Nome e Tema

Nome do jogo: Batalha Naval – Dois Jogadores

Temática: Duelo tático entre embarcações, onde cada jogador tenta afundar a frota inimiga.

## 2. Integrantes e Papéis

- Erick Vieira: Desenvolvedor Principal e Designer de Tabuleiro

- Patrick Lohn: Implementação de Lógica de Jogo e Pilha de Ações

- Rafael Sonoki: Controle de Turnos (Fila) e Interface de Botões

- Ruan: Animações e Estrutura de Código (Funções Auxiliares)

## 3. Descrição do Funcionamento

O jogo é um duelo de estratégia em que cada participante dispõe de um tabuleiro 10×10, escondendo sua própria frota de navios. Ao iniciar, todas as peças estão ocultas. Na sua vez, o jogador mira em uma coordenada do tabuleiro adversário clicando em uma célula.

* **Tiro e revelação**
  + Se o disparo acertar um navio, a célula revela a peça atingida e dispara uma animação de explosão.
  + Se errar, a célula simplesmente se abre em cor de fundo, indicando água.
* **Contador de tiros**  
  O total de disparos realizados é exibido em tempo real no canto superior.
* **Alternância de turnos**  
  Cada acerto ou erro consome o turno: após clicar, a vez passa automaticamente para o oponente.
* **Condição de vitória**  
  O jogo termina quando todas as partes de todos os navios de um dos jogadores forem atingidas. O vencedor é anunciado juntamente com o número total de tiros que disparou.

## 4. Uso de Estruturas de Dados

* **Listas**
  + **Tabuleiros 10×10**: cada tabuleiro é uma lista de listas, permitindo acesso direto à célula (x, y) em O(1) e inicialização simples via compreensões.
  + **Navios disponíveis**: armazenados em uma lista de strings, facilitando iteração para posicionamento e desenho.
* **Fila (collections.deque)**
  + **Turnos**: turn\_queue = deque(players) mantém a ordem de jogada. O método deque.rotate(-1) desloca o jogador atual para o fim em O(1), garantindo troca de turno instantânea e sem realocação dos elementos.
* **Pilha (lista Python)**
  + **Histórico de ações**: cada tiro é pushado como tupla (jogador, coordenada). Usamos append() e pop() em O(1), abrindo possibilidade de **undo** de forma trivial.

**Mapeamento via dicionários**  
Também usamos dicionários para associar cada jogador ao seu próprio tabuleiro e estado de revelação, agrupando estruturas relacionadas de forma clara.

# 5. Análise de Complexidade (Big O)

| **Operação** | **Estrutura** | **Tempo** | **Justificativa** |
| --- | --- | --- | --- |
| Rotação de turnos | deque.rotate() | **O(1)** | Ajusta internamente ponteiros do deque, sem percorrer ou copiar elementos. |
| Inserção/remoção no histórico | stack.append()/pop() | **O(1)** | Empilha/desempilha o último elemento sem deslocamento de memória. |
| Verificação de vitória (10×10) | Lista de listas | **O(n·m)** ⇒ **O(n²)** | Precisa examinar cada célula do tabuleiro para confirmar que não há navios não revelados restantes. |

Por que analisamos deque.rotate()?

A alternância de turnos é a operação **mais frequente e crítica** no fluxo de jogo: a cada tiro, a vez passa de um jogador para o outro. Usamos collections.deque porque sua rotação (deque.rotate(-1)) ajusta internamente ponteiros em **O(1)**, sem percorrer ou copiar elementos. Em contraste, operações como a verificação de vitória naturalmente exigem percorrer todo o tabuleiro (10×10), resultando em **O(n²)**. Essa escolha evidencia o benefício de usar uma **fila (deque)** para turnos.

**6. Instruções de Execução**

1. **Instale o Python 3.8+**
2. **Instale o Pygame**
3. pip install pygame
4. **Abra o terminal** (CMD ou PowerShell no Windows, Terminal no macOS/Linux) e navegue até a pasta do projeto:
5. cd /caminho/para/Battleship
6. **Execute o jogo**
7. python battleship.py

Se tiver múltiplas versões do Python, use

py battleship.py

— isso abrirá a janela gráfica do jogo, onde você poderá clicar em **NOVO JOGO** para reiniciar e **AJUDA** para consultar instruções durante a partida.