Renan Balbino de Medeiros

Prof. Fernando Marques Figueira Filho

Introdução às Técnicas de Programação

15 de novembro de 2025

**Relatório técnico**

Este projeto tem como objetivo o desenvolvimento de um jogo **Campo Minado** na linguagem de programação C, empregando as técnicas e conceitos aprendidos ao longo da disciplina. A primeira etapa, focada em vetores e mapeamento, estabeleceu a base funcional do jogo. Esta segunda entrega detalha a evolução do projeto, incorporando os conteúdos da Unidade 2, que introduziram um nível superior de complexidade e flexibilidade à aplicação, notadamente através do uso de **ponteiros**, **alocação dinâmica de memória** e **manipulação avançada de matrizes e strings**.

Além do aspecto técnico, o projeto continua a aliar prática e motivação, oferecendo uma abordagem pedagógica que valoriza o aprendizado por meio da experimentação. A implementação de funcionalidades como o menu inicial, a alocação dinâmica do tabuleiro e do nome do jogador, e a função de **flood fill** (abertura automática de casas seguras) permitiu aplicar diretamente os conceitos mais abstratos da linguagem C, como a aritmética de ponteiros e o gerenciamento de memória, em um contexto lúdico e interativo.

## ANÁLISE TÉCNICA

Nesta seção, apresenta-se a análise técnica do projeto em sua segunda fase, detalhando a metodologia, as ferramentas e a aplicação dos novos conceitos trabalhados na disciplina. O objetivo é evidenciar como os fundamentos teóricos da Unidade 2 foram aplicados de forma prática para expandir e refatorar o jogo Campo Minado.

**Metodologia**

O desenvolvimento foi mantido em ambiente Linux, utilizando as mesmas ferramentas de base da entrega anterior (GCC, Visual Studio Code). Houve, no entanto, uma expansão significativa das bibliotecas padrão utilizadas para dar suporte às novas funcionalidades:

- **Bibliotecas**: Além de *stdio.h*, *stdlib.h*, *time.h* e c*type.h*, foi incluída a biblioteca *string.h*. O uso de *stdlib.h* tornou-se crucial para as funções de alocação dinâmica (*malloc* ,*free*), enquanto *string.h* foi essencial para a implementação do menu e gerenciamento do nome do jogador.

**Aplicação dos Conceitos da U2**

A transição da Unidade 1 para a Unidade 2 representou um salto da programação estática para a dinâmica, focando no gerenciamento explícito de memória e na manipulação de estruturas de dados mais complexas.

**Ponteiros e Alocação Dinâmica**

O conceito mais impactante desta unidade foi a aplicação de ponteiros para a alocação dinâmica do tabuleiro. Na versão anterior, o tabuleiro era limitado por constantes de compilação (#define TAM). Nesta nova implementação, as matrizes tabuleiro e visivel são declaradas como ponteiros para ponteiros (ex: int **tabuleiro).

Isso exigiu um processo de alocação em duas etapas: primeiro, aloca-se um vetor de ponteiros (para as linhas) e, em seguida, utiliza-se uma estrutura de repetição para alocar cada linha individualmente (as colunas). Esta abordagem não só demonstra o uso avançado de ponteiros, como também torna o jogo flexível, permitindo que o tamanho do tabuleiro seja definido em tempo de execução, através do novo menu inicial.

A estratégia de implementação optou pelo uso desses ponteiros (tabuleiro e visivel) como variáveis globais. Isso permitiu que a maioria das funções de gerenciamento do jogo (como imprimir(), inicializar(), calcular_perigo() e liberar_memoria()) acessem diretamente a memória alocada sem a necessidade de alterar suas assinaturas para receber ponteiros duplos.

No entanto, o conceito de passagem por referência foi aplicado de forma crucial na função floodFill(). Esta função recursiva recebe explicitamente os ponteiros para as matrizes (int **visivel, int **vizinhos) como parâmetros. Isso permite que a função manipule diretamente o estado de visibilidade do tabuleiro, demonstrando uma aplicação prática e complexa da passagem de ponteiros em algoritmos matriciais.

**Strings e Funções da string.h**

Os conceitos de strings foram aplicados na captura e manipulação do nome do jogador. Para esta funcionalidade, foi utilizado um vetor de caracteres com tamanho estático (char nome[20]) na função main.

Embora a alocação não tenha sido dinâmica, a manipulação da string exigiu o uso de funções da biblioteca string.h. Especificamente, a função strcspn foi empregada após a leitura do nome com fgets. Como fgets armazena o caractere de nova linha (\n) no buffer, strcspn foi utilizada para localizar esse caractere e substituí-lo pelo terminador nulo (\0). Este tratamento foi essencial para garantir que o nome do jogador fosse exibido corretamente em mensagens futuras, como na tela de vitória, demonstrando o uso de funções de biblioteca para a correta manipulação de strings.

**Matrizes e Estruturas de Repetição Aninhadas**

Embora as matrizes já fossem a estrutura de dados central, sua manipulação foi aprimorada. A implementação da funcionalidade de **flood fill** (abertura automática) é a principal aplicação de **percorrimento bidimensional** avançado.

Quando o jogador seleciona uma célula com valor '0' (sem bombas vizinhas), o algoritmo é disparado. Ele utiliza **estruturas de repetição aninhadas** (implementadas de forma recursiva) para "visitar" todas as oito células vizinhas. Se uma célula vizinha também for '0', o processo se repete recursivamente; se for um número, a célula é apenas revelada. Esta lógica é um exemplo

clássico de "aplicação em problemas matriciais", exigindo um controle rigoroso dos índices e das condições de parada para evitar o acesso a posições inválidas da matriz.

Como funcionalidade bônus, um **timer** foi adicionado, utilizando a biblioteca *time.h* para capturar o tempo inicial (*time_t inicio*) e calcular o tempo decorrido ao final do jogo.

## IMPLEMENTAÇÃO E REFLEXÃO

As dificuldades técnicas desta segunda etapa foram significativamente diferentes das encontradas na Unidade 1. O principal desafio foi a transição do pensamento estático para o dinâmico, centrado no **gerenciamento de memória**. Compreender a sintaxe de ponteiros para ponteiros (int **) e, principalmente, garantir a correta liberação da memória (*free*) foi um obstáculo crucial. Foi necessário implementar rotinas de limpeza que percorrem a matriz com um loop, liberando cada linha individualmente antes de liberar o ponteiro principal (o vetor de linhas), evitando assim **memory leaks**.

Outro ponto complexo foi a implementação do **flood fill**. A lógica recursiva para o "percorrimento bidimensional" exigiu depuração cuidadosa para tratar os casos de borda (células nas extremidades do tabuleiro) e para garantir que a recursão parasse corretamente, sem entrar em loops infinitos ou acessar memória fora dos limites da matriz.

A **manipulação de strings** também apresentou seus desafios. A alocação dinâmica do nome do jogador exigiu a compreensão de como *string.h* interage com a memória, especialmente no que tange à necessidade de alocar espaço para o caractere nulo terminador (`\0`).

Em termos de aprendizado, o projeto consolidou de forma prática os conceitos de ponteiros, que são fundamentais na linguagem C. A capacidade de criar estruturas de dados cujo tamanho é definido pelo usuário, e não pelo programador, foi o ganho mais significativo desta etapa. O código tornou-se mais robusto, flexível e alinhado às práticas de programação que otimizam o uso de recursos.

Como melhorias futuras, planeja-se implementar o uso de "bandeiras" (flags) para que o jogador possa marcar células suspeitas, uma funcionalidade essencial do Campo Minado clássico. Além disso, considera-se a criação de um sistema de ranking e a possibilidade de salvar e carregar o estado atual do jogo, o que introduzirá novos desafios relacionados à manipulação de arquivos.