Técnicas e Desenvolvimento de Algoritmos

Relatório – Jogo Da Velha (Tic-Tac-Toe)

Nome e RGM dos integrantes da equipe:

Italo Henrique Cavalcante De Jesus – RGM: 37090089

Miguel Araújo Cavalcante Da Fonsêca – RGM: 38225808

João Victor de Figueiredo Abrósio – RGM: 38594552

Ian Lucas Guedes Rodrigues – RGM: 38507579

Introdução:

Descrição do jogo e regras:

O Jogo da Velha é um clássico jogo de tabuleiro para dois jogadores, onde o objetivo é formar uma linha com três símbolos iguais, seja na horizontal, vertical ou diagonal. No jogo desenvolvido, há dois modos disponíveis:

1. Jogador VS Computador: O jogador compete contra um adversário controlado pelo programa.
2. Jogador VS Jogador: Dois jogadores alternam turnos para competir.

Cada jogador deve escolher uma célula vazia do tabuleiro 3x3 para marcar com seu símbolo: **"X"** ou **"O"**. O jogo termina quando um jogador completa uma linha de três símbolos ou quando todas as células são preenchidas, resultando em empate.

O jogo também implementa um sistema de ranking para registrar as vitórias dos jogadores, armazenando-as em um arquivo para referência futura.

Resultados:

Descrição Geral Do Jogo:

O projeto foi desenvolvido em linguagem C, utilizando conceitos de manipulação de matrizes, leitura e gravação de arquivos, e geração de números aleatórios para jogadas do computador. As principais funcionalidades incluem:

- Inicialização dinâmica do tabuleiro.
- Verificação de vitórias ou empates.
- Sistema de ranking persistente.
- Interface textual amigável para interação com os jogadores.

Estrutura do tabuleiro: O tabuleiro foi projetado como uma matriz dinâmica de tamanho 3x3, permitindo flexibilidade para possíveis ajustes futuros, como ampliar o tamanho do jogo. A estrutura do código torna o tabuleiro visualmente intuitivo, com separadores e espaços claros entre as células, proporcionando uma boa experiência para os jogadores.

Verificação de Condições de Jogo: O sistema verifica automaticamente as condições de vitória ou empate após cada jogada. A verificação considera:

- Linhas completas: Todas as células de uma linha preenchidas pelo mesmo jogador.
- Colunas completas: Todas as células de uma coluna preenchidas pelo mesmo jogador.
- Diagonais completas: Todas as células de uma diagonal preenchidas pelo mesmo jogador.
- Empate: Quando o tabuleiro está completamente preenchido e nenhum jogador alcançou as condições de vitória.

Sistema de Ranking: Para tornar o jogo mais competitivo, foi implementado um sistema de ranking persistente que registra as vitórias de cada jogador em um arquivo de texto.

- Sempre que um jogador vence, o programa atualiza o arquivo. Se o jogador já existia no ranking, suas vitórias são incrementadas. Caso contrário, um novo registro é criado.
- Isso possibilita que jogadores acompanhem seu progresso e comparações futuras, mesmo após encerrar o programa.

Exemplificação do Código Fonte:

1. Inicialização do tabuleiro:
   Aloca dinamicamente memória para um tabuleiro 3x3
   Usa malloc() para alocação flexível de memória

```c
// inicializa o tabuleiro - malloc (alocação dinamica)
char** inicializarTabuleiro() {

    char** tabuleiro = (char**)malloc(TAMANHO * sizeof(char*));

    // Inicializa tudo em branco
    for (int i = 0; i < TAMANHO; i++) {
        tabuleiro[i] = (char*)malloc(TAMANHO * sizeof(char));
        for (int j = 0; j < TAMANHO; j++) {
            tabuleiro[i][j] = ' ';  // espaço em branco
        }
    }
    return tabuleiro;
}
```

2. Jogada do computador:
   Usa rand() para gerar coordenadas aleatórias
   Garante que só marca células vazias

```c
//loopcomputador: só joga em lugares livres
void jogadaComputador(char** tabuleiro) {
    int linha, coluna;
    do {
        linha = rand() % TAMANHO;
        coluna = rand() % TAMANHO;
    } while (tabuleiro[linha][coluna] != ' ');  //vazio

    tabuleiro[linha][coluna] = 'O';  //jogada
}
```

3. Sistema de Ranking:
   Utiliza arquivo texto para persistência do ranking
   Reescreve arquivo com dados atualizados

```c
// ranking txt
void atualizarRanking(const char* nome) {
    FILE* arquivo = fopen("ranking.txt", "a+");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo de ranking!\n"); // mensagem de erro
        return;
    }

    Jogador jogadores[100];
    int numJogadores = 0;
    int encontrado = 0;

    //jogadores já existentes
    while (fscanf(arquivo, "%s %d", jogadores[numJogadores].nome,
                &jogadores[numJogadores].vitorias) == 2) {
        // add vitórias a esses jogadores
        if (strcmp(jogadores[numJogadores].nome, nome) == 0) {
            jogadores[numJogadores].vitorias++;
            encontrado = 1;
        }
        numJogadores++;
    }

    // caso o jogador não esteja cadastrado
    if (!encontrado) {
        strcpy(jogadores[numJogadores].nome, nome);
        jogadores[numJogadores].vitorias = 1;
        numJogadores++;
    }

    // atualiza ranking
    fclose(arquivo);
    arquivo = fopen("ranking.txt", "w");

    for (int i = 0; i < numJogadores; i++) {
        fprintf(arquivo, "%s %d\n", jogadores[i].nome, jogadores[i].vitorias);
    }

    fclose(arquivo);
}

// exibe o arquivo atualizado com as informações que foram obtidas
void mostrarRanking() {
    FILE* arquivo = fopen("ranking.txt", "r");
    if (arquivo == NULL) {
        printf("Nenhum ranking disponível ainda!\n");
```

4. Verificação de Vitórias:
   Retorna 1 (verdadeiro) se encontrar 3 símbolos iguais
   Retorna 0 (falso) se não encontrar combinação vencedora

```c
int verificarVencedor(char** tabuleiro, char jogador) {
    // Linhas
    for (int i = 0; i < TAMANHO; i++) {
        if (tabuleiro[i][0] == jogador &&
            tabuleiro[i][1] == jogador &&
            tabuleiro[i][2] == jogador) {
            return 1;  // Linha completa
        }
    }

    //colunas
    for (int j = 0; j < TAMANHO; j++) {
        if (tabuleiro[0][j] == jogador &&
            tabuleiro[1][j] == jogador &&
            tabuleiro[2][j] == jogador) {
            return 1;  // Coluna completa
        }
    }

    //diagonais
    if ((tabuleiro[0][0] == jogador &&
         tabuleiro[1][1] == jogador &&
         tabuleiro[2][2] == jogador) ||
        (tabuleiro[0][2] == jogador &&
         tabuleiro[1][1] == jogador &&
         tabuleiro[2][0] == jogador)) {
        return 1;  // Diagonal completa
    }

    return 0;  // Sem um vencedor ainda
}
```

Dificuldades Encontradas e Soluções Implementadas:

Gerenciamento da memória dinâmica: Inicialmente, houve dificuldades em liberar a memória corretamente. A solução foi implementar a função liberarTabuleiro, que libera as linhas da matriz e o ponteiro principal, prevenindo vazamentos de memória. Uso de malloc() e free() para alocação dinâmica

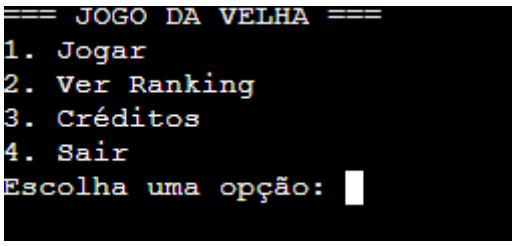
Validação das jogadas: Durante o desenvolvimento, foram encontrados erros ao validar as coordenadas de entrada dos jogadores. Isso foi resolvido com condições adicionais para verificar se a posição era válida e estava vazia.

Persistência no sistema de ranking: Garantir que o ranking fosse atualizado corretamente em cada partida exigiu ajustes na leitura e gravação do arquivo. O uso de um vetor de estruturas para armazenar temporariamente os dados foi fundamental para resolver o problema.

Demonstrativo das funcionalidades implementadas:

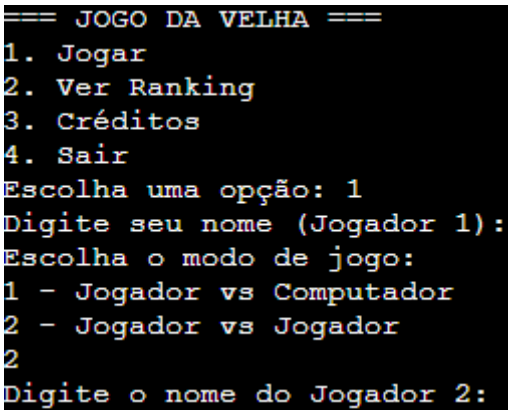
Tela inicial do jogo:

A tela inicial possibilita que o usuário inicie uma nova partida, veja o ranking, caso já tenha jogado alguma partida antes, veja os créditos e encerre o processo, tudo com uma interface intuitiva para que não haja complicações.



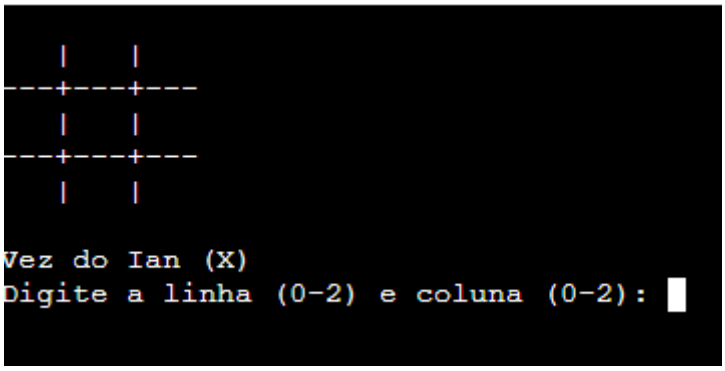
Menu de escolha de modo de jogo:

O jogo pede que o usuário escolha entre o modo PvP ou PvC, após isso coleta o nome do usuário, que posteriormente será armazenado no arquivo de ranking, caso o jogador em questão venha a ganhar a partida. A solicitação do nome do Player 2 virá apenas após a confirmação de que o modo de jogo escolhido seja o PvP



Jogada do usuário e interface do tabuleiro:

O tabuleiro inicial encontra-se vazio e preenche-se a cada movimento optado pelo jogador, seguido de uma jogada do computador ou do Player 2, instruindo o jogador a selecionar uma posição com base no número de linhas e colunas, respectivamente

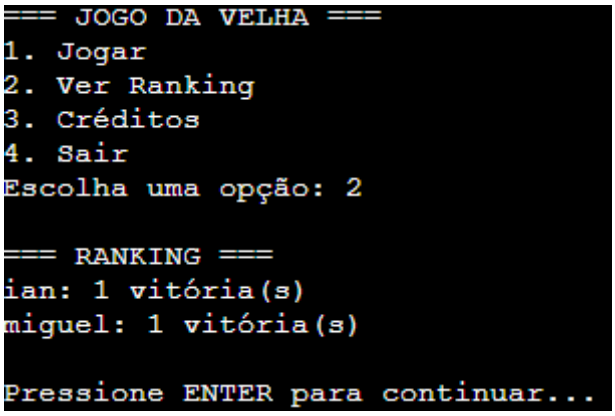


Ranking:

O ranking mostra as estatísticas de vitórias acumuladas por jogadores em partidas anteriores.

O sistema persiste os dados em um arquivo de texto, permitindo que as estatísticas sejam recuperadas em execuções futuras do jogo.

O arquivo armazena tanto as vitórias do P1 como do P2



Apêndice – Código Fonte:

```
#include <stdio.h>   // Biblioteca padrão de entrada/saída
#include <stdlib.h>  // Para funções como malloc, free, system
#include <string.h>  // Para manipulação de strings
#include <time.h>    // Para gerar números aleatórios baseados no tempo

#define TAMANHO 3    // Define o tamanho do tabuleiro como 3x3 (para facilitar futuras modificações)
```

```
// Estrutura para armazenar informações dos jogadores no ranking
// Permite que o jogo mantenha um registro das vitórias de cada jogador
typedef struct {
    char nome[50];   // Nome do jogador
    int vitorias;    // Número de vitórias deste jogador
} Jogador;

// Função para limpar a tela de forma multiplataforma
// Usa "cls" no Windows e "clear" em sistemas Unix-like
void limparTela() {
    // Verifica se está rodando no Windows e usa o comando apropriado
    #ifdef _WIN32
        system("cls");   // Limpa tela no Windows
    #else
        system("clear"); // Limpa tela em sistemas Unix/Linux
    #endif
}

// Função que cria e inicializa o tabuleiro do jogo
// Aloca memória dinamicamente para permitir flexibilidade
char** inicializarTabuleiro() {
    // Aloca memória para linhas do tabuleiro
    char** tabuleiro = (char**)malloc(TAMANHO * sizeof(char*));

    // Inicializa cada célula com espaço em branco
    for (int i = 0; i < TAMANHO; i++) {
        tabuleiro[i] = (char*)malloc(TAMANHO * sizeof(char));
        for (int j = 0; j < TAMANHO; j++) {
            tabuleiro[i][j] = ' ';  // Célula vazia
        }
    }
    return tabuleiro;
}
```

```
// Libera a memória alocada para o tabuleiro
// Importante para evitar vazamento de memória
void liberarTabuleiro(char** tabuleiro) {
    for (int i = 0; i < TAMANHO; i++) {
        free(tabuleiro[i]);  // Libera cada linha
    }
    free(tabuleiro);  // Libera o ponteiro principal
}

// Imprime o tabuleiro de forma visualmente agradável
void imprimirTabuleiro(char** tabuleiro) {
    printf("\n");
    for (int i = 0; i < TAMANHO; i++) {
        // Imprime cada linha com separadores verticais
        printf(" %c | %c | %c \n", tabuleiro[i][0], tabuleiro[i][1], tabuleiro[i][2]);

        // Adiciona separadores horizontais entre as linhas
        if (i < TAMANHO - 1) {
            printf("---+---+---\n");
        }
    }
    printf("\n");
}

// Verifica se há um vencedor
// Checa linhas, colunas e diagonais
int verificarVencedor(char** tabuleiro, char jogador) {
    // Verifica linhas
    for (int i = 0; i < TAMANHO; i++) {
        if (tabuleiro[i][0] == jogador &&
            tabuleiro[i][1] == jogador &&
            tabuleiro[i][2] == jogador) {
            return 1;  // Linha completa
        }
```

```
    }

    // Verifica colunas
    for (int j = 0; j < TAMANHO; j++) {
        if (tabuleiro[0][j] == jogador &&
            tabuleiro[1][j] == jogador &&
            tabuleiro[2][j] == jogador) {
            return 1;  // Coluna completa
        }
    }

    // Verifica diagonais
    if ((tabuleiro[0][0] == jogador &&
         tabuleiro[1][1] == jogador &&
         tabuleiro[2][2] == jogador) ||
        (tabuleiro[0][2] == jogador &&
         tabuleiro[1][1] == jogador &&
         tabuleiro[2][0] == jogador)) {
        return 1;  // Diagonal completa
    }

    return 0;  // Sem vencedor
}

// Verifica se o tabuleiro está completamente preenchido
int tabuleiroCompleto(char** tabuleiro) {
    for (int i = 0; i < TAMANHO; i++) {
        for (int j = 0; j < TAMANHO; j++) {
            if (tabuleiro[i][j] == ' ') {
                return 0;  // Ainda há células vazias
            }
        }
    }
    return 1;  // Tabuleiro cheio
```

```
}

// Função de jogada do computador
// Escolhe uma posição aleatória livre
void jogadaComputador(char** tabuleiro) {
    int linha, coluna;
    do {
        // Gera coordenadas aleatórias
        linha = rand() % TAMANHO;
        coluna = rand() % TAMANHO;
    } while (tabuleiro[linha][coluna] != ' ');  // Repete até encontrar célula vazia

    tabuleiro[linha][coluna] = 'O';  // Marca posição do computador
}

// Atualiza o ranking de vitórias
void atualizarRanking(const char* nome) {
    // Abre arquivo em modo append/leitura
    FILE* arquivo = fopen("ranking.txt", "a+");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo de ranking!\n");
        return;
    }

    Jogador jogadores[100];  // Limite de 100 jogadores
    int numJogadores = 0;
    int encontrado = 0;

    // Lê jogadores existentes
    while (fscanf(arquivo, "%s %d", jogadores[numJogadores].nome,
                  &jogadores[numJogadores].vitorias) == 2) {
        // Se jogador já existe, incrementa vitórias
        if (strcmp(jogadores[numJogadores].nome, nome) == 0) {
            jogadores[numJogadores].vitorias++;
```

```
            encontrado = 1;
        }
        numJogadores++;
    }

    // Adiciona novo jogador se não existir
    if (!encontrado) {
        strcpy(jogadores[numJogadores].nome, nome);
        jogadores[numJogadores].vitorias = 1;
        numJogadores++;
    }

    // Reescreve arquivo com dados atualizados
    fclose(arquivo);
    arquivo = fopen("ranking.txt", "w");

    for (int i = 0; i < numJogadores; i++) {
        fprintf(arquivo, "%s %d\n", jogadores[i].nome, jogadores[i].vitorias);
    }

    fclose(arquivo);
}

// Mostra o ranking de jogadores
void mostrarRanking() {
    FILE* arquivo = fopen("ranking.txt", "r");
    if (arquivo == NULL) {
        printf("Nenhum ranking disponível ainda!\n");
        return;
    }

    printf("\n=== RANKING ===\n");
    char nome[50];
    int vitorias;
```

```
    // Imprime cada jogador e suas vitórias
    while (fscanf(arquivo, "%s %d", nome, &vitorias) == 2) {
        printf("%s: %d vitória(s)\n", nome, vitorias);
    }

    fclose(arquivo);
}

// Função para mostrar os créditos do jogo
void mostrarCreditos() {
    printf("\n=== CRÉDITOS ===\n");
    printf("Jogo da Velha v1.0\n");
    printf("Desenvolvido por: Ian, Miguel, Italo, João Victor\n");
    printf("Data: 2024\n");
}

// Função principal do jogo
// Função principal do jogo
void jogar() {
    // Inicializa tabuleiro
    char** tabuleiro = inicializarTabuleiro();
    char nome1[50];  // Nome do primeiro jogador
    char nome2[50];  // Nome do segundo jogador
    int modo;

    // Captura nome e modo de jogo
    printf("Digite seu nome (Jogador 1): ");
    scanf("%s", nome1);

    // Escolha entre jogar contra computador ou outro jogador
    printf("Escolha o modo de jogo:\n");
    printf("1 - Jogador vs Computador\n");
    printf("2 - Jogador vs Jogador\n");
```

```
scanf("%d", &modo);

// Se for modo Jogador vs Jogador, captura nome do segundo jogador
if (modo == 2) {
    printf("Digite o nome do Jogador 2: ");
    scanf("%s", nome2);
}

int jogadorAtual = 1;  // Começa pelo jogador 1
int linha, coluna;

// Loop principal do jogo
while (1) {
    limparTela();
    imprimirTabuleiro(tabuleiro);

    // Vez do Jogador 1
    if (jogadorAtual == 1) {
        printf("Vez do %s (X)\n", nome1);
        printf("Digite a linha (0-2) e coluna (0-2): ");
        scanf("%d %d", &linha, &coluna);

        // Valida jogada
        if (linha >= 0 && linha < TAMANHO &&
            coluna >= 0 && coluna < TAMANHO &&
            tabuleiro[linha][coluna] == ' ') {
            tabuleiro[linha][coluna] = 'X';

            // Verifica vitória
            if (verificarVencedor(tabuleiro, 'X')) {
                limparTela();
                imprimirTabuleiro(tabuleiro);
                printf("Parabéns %s! Você venceu!\n", nome1);
                atualizarRanking(nome1);
```

```
            break;
        }

        jogadorAtual = 2;  // Próximo jogador
    } else {
        printf("Jogada inválida! Tente novamente.\n");
        getchar(); getchar();  // Pausa
        continue;
    }
} else {
    // Vez do Jogador 2 ou Computador
    if (modo == 1) {
        jogadaComputador(tabuleiro);
    } else {
        printf("Vez do %s (O)\n", nome2);
        printf("Digite a linha (0-2) e coluna (0-2): ");
        scanf("%d %d", &linha, &coluna);

        // Valida jogada
        if (linha >= 0 && linha < TAMANHO &&
            coluna >= 0 && coluna < TAMANHO &&
            tabuleiro[linha][coluna] == ' ') {
            tabuleiro[linha][coluna] = 'O';
        } else {
            printf("Jogada inválida! Tente novamente.\n");
            getchar(); getchar();  // Pausa
            continue;
        }
    }

    // Verifica vitória do Jogador 2
    if (verificarVencedor(tabuleiro, 'O')) {
        limparTela();
        imprimirTabuleiro(tabuleiro);
```

```
            if (modo == 1) {
                printf("O computador venceu!\n");
            } else {
                printf("Parabéns %s! Você venceu!\n", nome2);
                atualizarRanking(nome2);
            }
            break;
        }

        jogadorAtual = 1;  // Volta para o Jogador 1
    }

    // Verifica empate
    if (tabuleiroCompleto(tabuleiro)) {
        limparTela();
        imprimirTabuleiro(tabuleiro);
        printf("Empate!\n");
        break;
    }
  }

  // Libera memória e pausa
  liberarTabuleiro(tabuleiro);
  printf("\nPressione ENTER para continuar...");
  getchar(); getchar();
}
// Função principal que gerencia o menu
int main() {
  // Inicializa gerador de números aleatórios
  srand(time(NULL));

  int opcao;

  // Menu de opções
```

```
    do {
        limparTela();
        printf("=== JOGO DA VELHA ===\n");
        printf("1. Jogar\n");
        printf("2. Ver Ranking\n");
        printf("3. Créditos\n");
        printf("4. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Processamento da escolha do usuário
        switch (opcao) {
            case 1: jogar(); break;
            case 2:
                mostrarRanking();
                printf("\nPressione ENTER para continuar...");
                getchar(); getchar();
                break;
            case 3:
                mostrarCreditos();
                printf("\nPressione ENTER para continuar...");
                getchar(); getchar();
                break;
            case 4:
                printf("Obrigado por jogar!\n");
                break;
            default:
                printf("Opção inválida!\n");
                break;
        }
    } while (opcao != 4);

    return 0;
}
```