

CENTRO UNIVERSITÁRIO DE JOÃO PESSOA – UNIPÊ
CIÊNCIA DA COMPUTAÇÃO 2024.2
TÉCNICAS E DESENVOLVIMENTO DE ALGORITMOS

Relatório de projeto final – Tic Tac Toe

Docente: Wallace Sartori Bonfim

Discentes:

- Beatriz Almeida de Souza Silva – RGM: 38500779
 - Gabriel Bianchini Carvalho – RGM: 38519623
 - José Carlos de Oliveira Neto – RGM: 38491516
 - Rigel Silva de Souza Sales – RGM: 37258346
-

1. INTRODUÇÃO

O Tic Tac Toe, também conhecido como "Jogo da Velha", é um jogo clássico em que dois jogadores competem para preencher uma grade 3x3 com seus símbolos (X ou O). O objetivo é formar uma linha com três símbolos iguais, seja na horizontal, vertical ou diagonal. O jogo termina quando um jogador vence ou quando o tabuleiro é preenchido sem vencedores, resultando em empate.

Regras:

- Dois jogadores se alternam para fazer suas jogadas.
 - Cada jogador escolhe uma célula vazia na grade 3x3.
 - O primeiro a alinhar três símbolos consecutivos vence.
 - O jogador vencedor recebe 10 pontos.
 - Caso todas as células sejam preenchidas e ninguém tenha vencido, o jogo termina em empate.
-

2. RESULTADOS

2.1 DESCRIÇÃO

O projeto foi implementado em linguagem C e rodado no console.

Funcionalidades:

- Um tabuleiro 3x3 exibido em cada turno.
- Alternância entre dois jogadores, verificando automaticamente vitória ou empate.
- Funcionalidade adicional para jogar contra o computador.
- Registro de partidas no ranking salvo em um arquivo.

2.2 DIFICULDADES ENCONTRADAS E SOLUÇÕES

- **Assegurar que as entradas seriam um único dígito:** Para isso, foi implementada a função `validarEntrada()`.

```
// FUNÇÕES PARA VERIFICAR SE A ENTRADA DO JOGADOR É COMPOSTA POR APENAS DÍGITOS E NÃO ESTÁ VAZIA
int validarEntrada(char *entrada) {
    int i = 0;

    if (entrada[0] == '\0') {
        return 0;
    }

    while (entrada[i] != '\0') {
        if (!isdigit(entrada[i])) {
            return 0;
        }
        i++;
    }
    return 1;
}
```

- **Erro na geração do arquivo de ranking:** Após a implementação das funções de ranking, o arquivo não estava sendo gerado, por problemas relacionados ao diretório-destino. Isso foi resolvido mudando-se o endereço no qual o arquivo seria criado.
- **Pontuações erradas no arquivo de ranking:** Ao gerar o arquivo de ranking, as pontuações estavam vindo como valores aleatórios alocados na memória, e não com o valor 10. Isso foi resolvido implementando-se linhas de código que inicializavam as pontuações de cada jogador com valor 0.

```
printf("Digite o nome do Jogador 'X': ");
fgets(jogador[0].nome, sizeof(jogador[0].nome), stdin);
jogador[0].simbolo = 'X';
jogador[0].nome[strlen(jogador[0].nome, "\n")] = '\0';
jogador[0].pontuacao = 0;

printf("Digite o nome do Jogador 'O': ");
fgets(jogador[1].nome, sizeof(jogador[1].nome), stdin);
jogador[1].simbolo = 'O';
jogador[1].nome[strlen(jogador[1].nome, "\n")] = '\0';
jogador[1].pontuacao = 0;
```

2.3 CAPTURAS DE TELA

```
PS C:\Users\biaal\Documents\GitHub\tic-tac-toe-2024.2> gcc src/main.c src/c
PS C:\Users\biaal\Documents\GitHub\tic-tac-toe-2024.2> src/main.exe
```

THE TOC TOC

*** TIC TAC TOE ***

1. Jogar
2. Ver Ranking
3. Creditos
4. Sair

Escolha uma opcao:

Iniciando jogo...

Digite o nome do Jogador 'X': Joao

Digite o nome do Jogador '0': Maria

Joao (X), escolha em qual linha jogar (1-3):

A 3x3 grid of dashed lines. The center cell contains a white 'X' on a black background.

A 3x3 grid with a cross pattern. The center cell contains the letter 'x' and the bottom-right cell contains the number '0'.

THE TOGO

CREDITOS
Beatriz Almeida de Souza Silva
Gabriel Bianchini Carvalho
Jose Carlos de Oliveira Neto
Rigel Silva de Souza Sales

```

  |  |
---|---|---
X | X |
---|---|---
  | 0 | 0

```

Joao (X), escolha em qual linha jogar (1-3): 2
 Joao (X), escolha em qual coluna jogar (1-3): 3

```

  |  |
---|---|---
X | X | X
---|---|---
  | 0 | 0

```

Parabens Joao, voce foi o vencedor!
 Arquivo de ranking não encontrado. Criando novo ranking...

```

1  ***TIC TAC TOE - Ranking:***
2
3  1. Joao - 10 pontos
4
5  Ultima atualização em: 28/11/2024 07:29:47 UTC-3

```

```

*** TIC TAC TOE ***
1. Jogar
2. Ver Ranking
3. Creditos
4. Sair

Escolha uma opcao: 2

Acessando ranking...

Ranking lido com sucesso!

***TIC TAC TOE - Ranking:***

1. Joao - 10 pontos

Ultima atualização em: 28/11/2024 07:29:47 UTC-3

```

```
*** TIC TAC TOE ***
1. Jogar
2. Ver Ranking
3. Creditos
4. Sair

Escolha uma opcao: 4

Saindo do jogo... Ate logo!
```

CÓDIGO FONTE

main.c:

```
#include <stdio.h>

#include <locale.h>

#include "load-game.h"
```

```
int main() {

    setlocale(LC_ALL, "pt_BR.UTF-8");

    mostrarLogo();

    menuInicial();

}
```

load-game.h:

```
// CARREGANDO AS FUNÇÕES
```

```
#ifndef LOAD_GAME_H
```

```
#define LOAD_GAME_H
```

```
// printing-functions.c:
```

```

void mostrarLogo();
void menuInicial();
void imprimeTabuleiro();
void mostrarCreditos();

// control-functions.c:
char iniciarJogo();
void realizarJogada(char tabuleiro[3][3], int linha, int
coluna, char simbolo);
int validarEntrada();
int validarPosicao();
int validarJogada();
int verificarVencedor();

// ranking-functions.c:
void escreveRanking();
void lerRanking();
void atualizarRanking(char *nome, int pontos);

#endif

printing-functions.c:

// FUNÇÕES PARA PRINTAR INTERFACE E OUTROS ELEMENTOS DO JOGO

#include <stdio.h> // Biblioteca de funções
entrada e saída (printf, scanf, etc)

```

```

#include <locale.h>                                // Biblioteca para
configuração de localidade e idioma

#include <string.h>                                // Biblioteca de funções
para string

#include <ctype.h>                                // Biblioteca para
manipulação de caracteres (toupper, isdigit, etc)

#include <stdlib.h>                                // Biblioteca de funções
para manipulação de memória

#include <windows.h>                                // Biblioteca específica
do Windows

#include <time.h>                                // Biblioteca de funções
de tempo

#include "load-game.h"                            // Arquivo declarando as
funções de ranking

```

```

// FUNÇÃO PARA PRINTAR A LOGO

```

```

void mostrarLogo() {

```

```

    printf("=====
=====\\n");

```

```

        printf("  -----  -          -----          -----
\\n");

```

```

        printf("  |_   _|(-)  ---  |_   _|__ -   ---  |_   _|___
---  \\n");

```

```

        printf("      | |   | | /  __|   | | /  _  | /  __|   | | /  _  \\
/  _  \\ \\ \\n");

```

```

        printf("      | |   | || (-__   | || (-| || (-__   | | |(-)| |
__/  \\n");

```

```

        printf("      |_|   |_|  \\ \\___|   |_|  \\ \\____|  \\ \\___|   |_|
\\ \\___/  \\ \\___|  \\n");

```



```
printf("
\n");

printf("=====
====\n");

    printf("\n");

    return;

};
```

```
// FUNÇÃO PARA MOSTRAR O MENU DE OPÇÕES DO JOGO
```

```
void menuInicial() {
    setlocale(LC_ALL, "pt_BR.UTF-8");

    int opcao = 0;
    char entrada[10];

    do {
        printf("*** TIC TAC TOE ***\n");

        printf("1. Jogar\n");
        printf("2. Ver Ranking\n");
        printf("3. Creditos\n");
        printf("4. Sair\n\n");
        printf("Escolha uma opcao: ");
```

```
fgets(entrada, sizeof(entrada), stdin);

entrada[strcspn(entrada, "\n")] = '\0';

if (!validarEntrada(entrada)) {
    printf("\nOpcao invalida, tente novamente.\n");
    continue;
}

opcao = atoi(entrada);

switch (opcao)
{
    case 1:
        printf("\nIniciando jogo...\n\n");
        iniciarJogo();
        break;

    case 2:
        printf("\nAcessando ranking...\n\n");
        lerRanking();
        break;

    case 3:
```

```

        mostrarCreditos();
        break;

    case 4:
        printf("\nSaindo do jogo... Ate logo! \n\n");
        break;

    default:
        printf("\nOpcao invalida, tente
novamente.\n");
        break;
    }
} while (opcao != 4);

return;
}

```

```

// FUNÇÃO PARA PRINTAR O TABULEIRO DO JOGO
void imprimeTabuleiro(char tabuleiro[3][3]) {
    printf("\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf(" %c ", tabuleiro[i][j]);
            if (j < 2) {

```

```
        printf("|");
    }

}

printf("\n");

if (i < 2) {
    printf("---|---|---\n");
}

}

printf("\n");
}
```

```
void mostrarCreditos() {
    // ARRAY DE STRINGS LITERAIS
    const char *autores[] = {
        "Beatriz Almeida de Souza Silva",
        "Gabriel Bianchini Carvalho",
        "Jose Carlos de Oliveira Neto",
        "Rigel Silva de Souza Sales"
    };

    int numAutores = sizeof(autores) / sizeof(autores[0]);

    int comprimentoMax = 0;
    for (int i = 0; i < numAutores; i++) {
        int comprimento = strlen(autores[i]);
```

```
        if (comprimento > comprimentoMax) {
            comprimentoMax = comprimento;
        }
    }

    comprimentoMax += 6;

    // PRINTANDO FORMATO DA CAIXA E TÍTULO
    printf("\n+");
    for (int i = 0; i < comprimentoMax; i++) {
        printf("-");
    }
    printf("+\n");

    const char *titulo = "CREDITOS";
    int espacosTitulo = (comprimentoMax - strlen(titulo)) / 2;
    printf("|");
    for (int i = 0; i < espacosTitulo; i++) {
        printf(" ");
    }
    printf("%s", titulo);

    for (int i = 0; i < comprimentoMax - strlen(titulo) -
espacosTitulo; i++) {
        printf(" ");
    }
}
```

```
printf("|\\n");

printf("|");
for (int i = 0; i < comprimentoMax; i++) {
    printf("-");
}
printf("|\\n");

for (int i = 0; i < numAutores; i++) {
    int espacos = comprimentoMax - strlen(autores[i]) - 2;
    printf("| %s", autores[i]);
    for (int j = 0; j < espacos; j++) {
        printf(" ");
    }
    printf("|\\n");
}

printf("+");
for (int i = 0; i < comprimentoMax; i++) {
    printf("-");
}
printf("+\\n");

printf("\\n");
}
```

control-functions.c:

// FUNÇÕES QUE RODAM O JOGO E REALIZAM O TRATAMENTO DE ERROS

```
#include <stdio.h> // Biblioteca de funções  
entrada e saída (printf, scanf, etc)
```

```
#include <locale.h> // Biblioteca para  
configuração de localidade e idioma
```

```
#include <string.h> // Biblioteca de funções  
para string
```

```
#include <ctype.h> // Biblioteca para  
manipulação de caracteres (toupper, isdigit, etc)
```

```
#include <stdlib.h> // Biblioteca de funções  
para manipulação de memória
```

```
#include <windows.h> // Biblioteca específica  
do Windows
```

```
#include <time.h> // Biblioteca de funções  
de tempo
```

```
#include "load-game.h" // Arquivo declarando as  
funções de ranking
```

// FUNÇÃO PARA INICIAR A PARTIDA

```
char iniciarJogo() {  
    char entrada[5];  
    int linha;  
    int coluna;  
    int resultado = 0;  
    int turno = 0;
```

```
typedef struct {
    char nome[50];
    char simbolo;
    int pontuacao;
} Player;

char tabuleiro[3][3] = {

    {' ', ' ', ' '},
    {' ', ' ', ' '},
    {' ', ' ', ' '}

};

Player jogador[2];

printf("Digite o nome do Jogador 'X': ");
fgets(jogador[0].nome, sizeof(jogador[0].nome), stdin);
jogador[0].simbolo = 'X';
jogador[0].nome[strcspn(jogador[0].nome, "\n")] = '\0';
jogador[0].pontuacao = 0;

printf("Digite o nome do Jogador 'O': ");
fgets(jogador[1].nome, sizeof(jogador[1].nome),
stdin);
```



```
jogador[1].simbolo = 'O';
jogador[1].nome[strcspn(jogador[1].nome, "\n")] = '\0';
jogador[1].pontuacao = 0;

while(resultado == 0) {
    imprimeTabuleiro(tabuleiro);

    printf("%s (%c), escolha em qual linha jogar (1-3): ",
jogador[turno].nome, jogador[turno].simbolo);

    fgets(entrada, sizeof(entrada), stdin);
    entrada[strcspn(entrada, "\n")] = '\0';

    if (!validarEntrada(entrada)) {
        printf("\nEntrada invalida %s, digite um valor
entre 1 e 3.\n", jogador[turno].nome);
        continue;
    }

    linha = atoi(entrada);

    printf("%s (%c), escolha em qual coluna jogar (1-3):
", jogador[turno].nome, jogador[turno].simbolo);

    fgets(entrada, sizeof(entrada), stdin);
    entrada[strcspn(entrada, "\n")] = '\0';

    if (!validarEntrada(entrada)) {
```

```

        printf("\nEntrada invalida %s, digite um valor
entre 1 e 3.\n", jogador[turno].nome);

        continue;

    }

    coluna = atoi(entrada);

    if(validarJogada(tabuleiro, linha - 1, coluna - 1)) {

        realizarJogada(tabuleiro, linha - 1, coluna - 1,
jogador[turno].simbolo);

        resultado = verificarVencedor(tabuleiro);

        if(resultado == 1) {

            imprimeTabuleiro(tabuleiro);

            printf("Parabens %s, voce foi o vencedor!\n",
jogador[turno].nome);

            jogador[turno].pontuacao += 10;

            atualizarRanking(jogador[turno].nome,
jogador[turno].pontuacao);

            printf("\n\nPressione 'Enter' para voltar ao
Menu principal.");

            getchar();

            break;

        }
    }
}

```

```

        else if(resultado == -1)    {
            imprimeTabuleiro(tabuleiro);
            printf("O jogo terminou empatado.\n");
            printf("\n\nPressione 'Enter' para voltar ao
Menu principal.");
            getchar();
            break;
        }

        turno = 1 - turno;
    }

    else    {
        printf("\nJogada invalida %s, tente novamente.\n",
jogador[turno].nome);
    }
}

return 0;
}

```

// FUNÇÃO PARA ATUALIZAR O TABULEIRO COM OS SÍMBOLOS DOS JOGADORES NAS POSIÇÕES ESPECIFICADAS

```

void realizarJogada(char tabuleiro[3][3], int linha, int
coluna, char simbolo)    {
    tabuleiro[linha][coluna] = simbolo;
}

```

```
// FUNÇÕES PARA VERIFICAR SE A ENTRADA DO JOGADOR É COMPOSTA  
POR APENAS DÍGITOS E NÃO ESTÁ VAZIA
```

```
int validarEntrada(char *entrada) {  
    int i = 0;  
  
    if (entrada[0] == '\0') {  
        return 0;  
    }  
  
    while (entrada[i] != '\0') {  
        if (!isdigit(entrada[i])) {  
            return 0;  
        }  
        i++;  
    }  
    return 1;  
}
```

```
int validarPosicao(int *entrada) {  
    int i = 0;  
  
    if (entrada[0] == '\0') {  
        return 0;  
    }
```

```

    }

    while (entrada[i] != '\0') {
        if (!isdigit(entrada[i])) {
            return 0;
        }
        i++;
    }
    return 1;
}

// FUNÇÃO PARA VERIFICAR SE A JOGADA ESTÁ NOS LIMITES DO
// TABULEIRO E O ESPAÇO ESTÁ VAZIO
int validarJogada(char tabuleiro[3][3], int linha, int coluna)
{
    if(linha >= 0 && linha < 3 && coluna >= 0 && coluna <3) {
        if(tabuleiro[linha][coluna] == ' ') {
            return 1;
        }
    }

    return 0;
}

```

```
// FUNÇÃO PARA CHECAR SE HOUE VENCEDOR, EMPATE OU O JOGO  
CONTINUA
```

```
int verificarVencedor(char tabuleiro[3][3]) {  
    for(int i = 0; i < 3; i++) {  
        if (tabuleiro[i][0] == tabuleiro[i][1] &&  
tabuleiro[i][1] == tabuleiro[i][2] && tabuleiro[i][0] != ' ' )  
            return 1;  
  
        if (tabuleiro[0][i] == tabuleiro[1][i] &&  
tabuleiro[1][i] == tabuleiro[2][i] && tabuleiro[0][i] != ' ' )  
            return 1;  
  
    }  

```

```
    if (tabuleiro[0][0] == tabuleiro[1][1] && tabuleiro[1][1]  
== tabuleiro[2][2] && tabuleiro[0][0] != ' ' )  
        return 1;  
  
    if (tabuleiro[0][2] == tabuleiro[1][1] && tabuleiro[1][1]  
== tabuleiro[2][0] && tabuleiro[0][2] != ' ' )  
        return 1;  

```

```
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        if (tabuleiro[i][j] == ' ' )  
            return 0;  
    }  
}
```

```

    }

}

return -1;

}

```

ranking-functions.c:

// FUNÇÕES DE ESCREVER, LER E FAZER ATUALIZAÇÃO DO ARQUIVO DE RANKING

```

#include <stdio.h> // Biblioteca de funções
entrada e saída (printf, scanf, etc)

#include <locale.h> // Biblioteca para
configuração de localidade e idioma

#include <string.h> // Biblioteca de funções
para string

#include <ctype.h> // Biblioteca para
manipulação de caracteres (toupper, isdigit, etc)

#include <stdlib.h> // Biblioteca de funções
para manipulação de memória

#include <windows.h> // Biblioteca específica
do Windows

#include <time.h> // Biblioteca de funções
de tempo

#include "load-game.h" // Arquivo declarando as
funções de ranking

```

// FUNÇÃO DE CRIAÇÃO DO ARQUIVO DE RANKING

```
// Leitura e escrita do arquivo .txt

// Realiza a manipulação da memória para armazenar as
informações de ranking

void escreveRanking() {

    // STRUCT DE JOGADOR NO RANKING

    typedef struct {

        int rank;

        int pontos;

        char nome[50];

    } Jogador;

    int rank, pontos;

    char nome[50];

    int linhaAtual = 0;

    char linha[50];

    int capacidadeRanking = 15;

    int tamanhoRanking = 0;

    char dataHora[100];

    Jogador *ranking = (Jogador *)malloc(capacidadeRanking *
sizeof(Jogador));

    if(!ranking) {

        printf("Erro na alocação de memória.\n\n");

        return;

    }
```



```
FILE *fileRanking = fopen("./ranking.txt", "r+");

if(fileRanking == NULL)    {
    printf("Acesso ao arquivo ranking.txt não
realizado.\n\n");
    free(ranking);
    return;
}

printf("Acesso ao arquivo ranking.txt realizado com
sucesso.\n\n");

while(fgets(linha, sizeof(linha), fileRanking)) {
    if (linhaAtual < 2) {
        linhaAtual++;
        continue;
    }

    if (tamanhoRanking >= capacidadeRanking) {
        capacidadeRanking *= 2;

        ranking = (Jogador *)realloc(ranking,
capacidadeRanking * sizeof(Jogador));

        if (!ranking) {
            printf("Erro ao redimensionar memória.\n");
            fclose(fileRanking);
        }
    }
}
```

```

        return;
    }
}

Jogador temp;

    if (sscanf(linha, "%d. %[^-] - %d pontos", &temp.rank,
temp.nome, &temp.pontos) == 3) {
        ranking[tamanhoRanking] = temp;
        tamanhoRanking++;
    } else {
        printf("Encontrado formato de linha invalido.\n");
        printf("A seguinte linha não sera considerada para
o ranking: %s\n\n", linha);
    }

}

for (int i = 0; i < tamanhoRanking - 1; i++) {
    for (int j = 0; j < tamanhoRanking - i - 1; j++) {
        if (ranking[j].pontos < ranking[j + 1].pontos) {
            Jogador temp = ranking[j];
            ranking[j] = ranking[j + 1];
            ranking[j + 1] = temp;
        }
    }
}

```

```

    }

    for (int i = 0; i < tamanhoRanking; i++) {
        ranking[i].rank = i + 1;
    }

    fileRanking = fopen("./ranking.txt", "w");

    if (fileRanking == NULL) {
        printf("Erro ao abrir o arquivo 'ranking.txt' no modo
escrita.\n");
        free(ranking);
        return;
    }

    fprintf(fileRanking, "***TIC TAC TOE - Ranking:***\n\n");

    for (int i = 0; i < tamanhoRanking; i++) {
        fprintf(fileRanking, "%d. %s - %d pontos\n",
ranking[i].rank, ranking[i].nome, ranking[i].pontoss);
    }

    time_t horaAtual = time(NULL);
    struct tm *tempoLocal = localtime(&horaAtual);

```

```
    strftime(dataHora, sizeof(dataHora), "%d/%m/%Y %H:%M:%S",
tempoLocal);
```

```
    fprintf(fileRanking, "\nUltima aualizacao em: %s UTC-3",
dataHora);
```

```
    fclose(fileRanking);
```

```
    for (int i = 0; i < tamanhoRanking; i++) {
        printf("%d. %s - %d pontos\n", ranking[i].rank,
ranking[i].nome, ranking[i].pontos);
    }
```

```
    printf("\n");
}
```

```
// FUNÇÃO PARA LEITURA DO RANKING:
```

```
// Lê e exibe o conteúdo do arquivo ranking.txt no console.
```

```
void lerRanking() {
```

```
    setlocale(0, "Portuguese");
```

```
    FILE *fileRanking = fopen("./ranking.txt", "r");
```

```
    char linha[50];
```

```
    char nome[50];
```

```
int rank, pontos;

int linhaAtual = 0;

if (fileRanking == NULL) {
    printf("Erro ao abrir o arquivo.\n\n");
    return;
} else {
    printf("Ranking lido com sucesso!\n\n");
}

while(fgets(linha, sizeof(linha), fileRanking)) {
    printf("%s", linha);
}

fclose(fileRanking);

    printf("\n\nPressione 'Enter' para voltar ao Menu
principal.");
    getchar();

    printf("\n");
}

// FUNÇÃO PARA ATUALIZAÇÃO DO RANKING:
```

```
// Adiciona ou atualiza as informações de um jogador específico no ranking.
```

```
// Caso o jogador já exista, seus pontos são incrementados.
```

```
// Se for um jogador novo, ele é adicionado ao ranking.
```

```
void atualizarRanking(char *nome, int pontos) {
```

```
    typedef struct {
```

```
        int rank;
```

```
        int pontos;
```

```
        char nome[50];
```

```
    } Jogador;
```

```
    Jogador *ranking = NULL;
```

```
    int capacidadeRanking = 15;
```

```
    int tamanhoRanking = 0;
```

```
    char linha[100];
```

```
    char dataHora[100];
```

```
        ranking = (Jogador *)malloc(capacidadeRanking * sizeof(Jogador));
```

```
    if (!ranking) {
```

```
        printf("Erro na alocação de memória.\n");
```

```
        return;
```

```
    }
```

```
    FILE *fileRanking = fopen("./ranking.txt", "r");
```

```
    if (fileRanking == NULL) {
```

```

        printf("Arquivo de ranking não encontrado. Criando
novo ranking...\n");

        fileRanking = fopen("./ranking.txt", "w");

        if (fileRanking == NULL) {

            printf("Erro ao criar o arquivo
'ranking.txt'.\n");

            free(ranking);

            return;

        }

        fprintf(fileRanking, "***TIC  TAC  TOE  -
Ranking:***\n\n");

        fclose(fileRanking);

    } else {

        while (fgets(linha, sizeof(linha), fileRanking)) {

            if (tamanhoRanking >= capacidadeRanking) {

                capacidadeRanking *= 2;

                ranking = (Jogador *)realloc(ranking,
capacidadeRanking * sizeof(Jogador));

                if (!ranking) {

                    printf("Erro ao redimensionar
memória.\n");

                    fclose(fileRanking);

                    return;

                }

            }

        }

        Jogador temp;

```

```
        if (sscanf(linha, "%d. %[^-] - %d pontos",
&temp.rank, temp.nome, &temp.pontos) == 3) {

            ranking[tamanhoRanking++] = temp;

        }

    }

    fclose(fileRanking);
}
```

```
int jogadorEncontrado = 0;

for (int i = 0; i < tamanhoRanking; i++) {

    if (strcmp(ranking[i].nome, nome) == 0) {

        ranking[i].pontos += pontos;

        jogadorEncontrado = 1;

        break;

    }

}

if (!jogadorEncontrado) {

    if (tamanhoRanking >= capacidadeRanking) {

        capacidadeRanking *= 2;

        ranking = (Jogador *)realloc(ranking,
capacidadeRanking * sizeof(Jogador));

        if (!ranking) {

            printf("Erro ao redimensionar memória.\n");

            return;

        }

    }

}
```



```
        strncpy(ranking[tamanhoRanking].nome, nome,
sizeof(ranking[tamanhoRanking].nome));

        ranking[tamanhoRanking].pontoss = pontos;

        tamanhoRanking++;

    }
}
```

```
for (int i = 0; i < tamanhoRanking - 1; i++) {
    for (int j = 0; j < tamanhoRanking - i - 1; j++) {
        if (ranking[j].pontoss < ranking[j + 1].pontoss) {
            Jogador temp = ranking[j];
            ranking[j] = ranking[j + 1];
            ranking[j + 1] = temp;
        }
    }
}
```

```
fileRanking = fopen("./ranking.txt", "w");
if (fileRanking == NULL) {
    printf("Erro ao abrir o arquivo para escrita.\n");
    free(ranking);
    return;
}
```

```
fprintf(fileRanking, "***TIC TAC TOE - Ranking:***\n\n");
```

```
    for (int i = 0; i < tamanhoRanking; i++) {  
        fprintf(fileRanking, "%d. %s - %d pontos\n", i + 1,  
ranking[i].nome, ranking[i].pontoss);  
    }  
  
    time_t horaAtual = time(NULL);  
    struct tm *tempoLocal = localtime(&horaAtual);  
    strftime(dataHora, sizeof(dataHora), "%d/%m/%Y %H:%M:%S",  
tempoLocal);  
    fprintf(fileRanking, "\nUltima atualização em: %s UTC-3",  
dataHora);  
  
    fclose(fileRanking);  
    free(ranking);  
}
```