NOME E RGM

|  |  |
| --- | --- |
| GABRIEL VICTOR SILVA NEGROMONTE | 34287655 |
| JONNATHAN VITURIANO DA COSTA | 34629963 |
| LUIS HENRIQUE ARAUJO BELMONT | 34187588 |
| LUIS HENRIQUE ÂNCORES PAIVA | 34437151 |
| THIAGO FERNANDES VILAR OLIVEIRA | 34439587 |

INTRODUÇÃO

NÚMEROS SUSPE**¿**TOS

O jogo da velha é um clássico passatempo que envolve dois jogadores, marcando alternadamente as posições em um tabuleiro 3x3 ou 5x5. O objetivo é formar uma linha contínua de três (ou mais, dependendo do tamanho do tabuleiro) de suas próprias marcas (X ou O), seja horizontal, vertical ou diagonal. O jogo termina quando um jogador consegue essa façanha ou quando o tabuleiro é preenchido sem que nenhum jogador vença, resultando em empate.

RESULTADOS

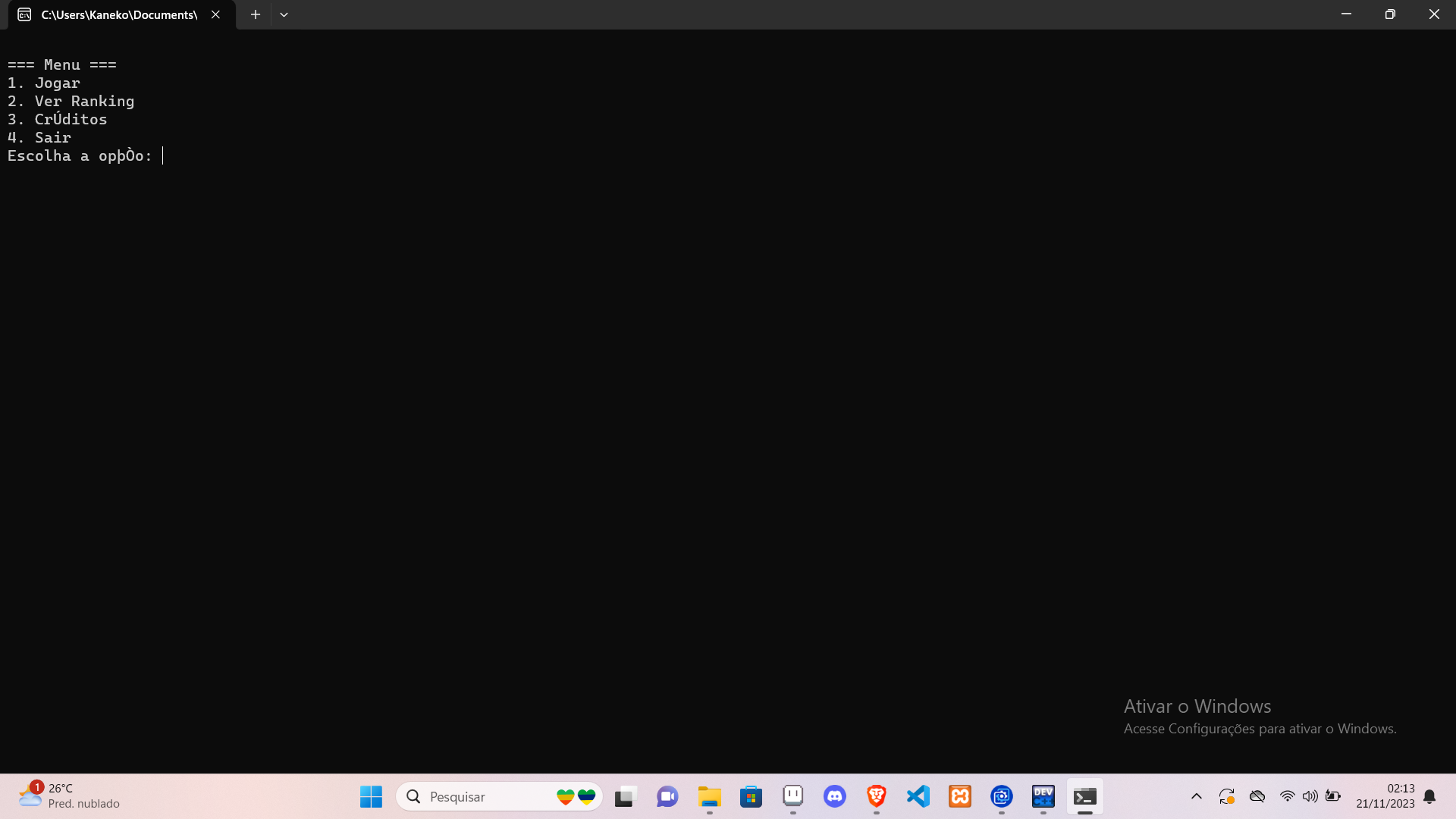
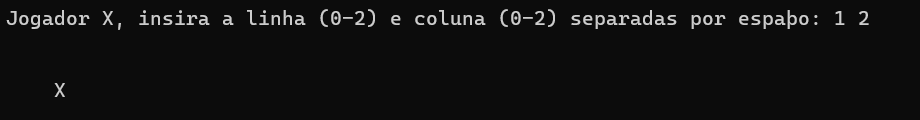
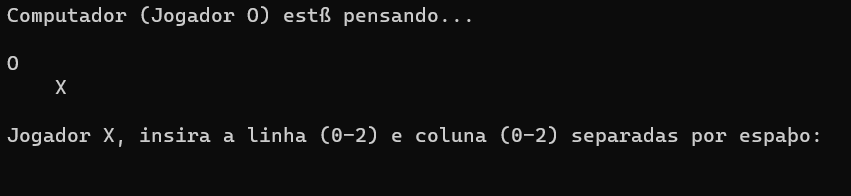
O código-fonte abaixo representa uma implementação simples do jogo da velha em linguagem C, atendendo a requisitos específicos, como um menu, jogabilidade contra o computador, sistema de ranking e alocação dinâmica de memória.

**Dificuldades Encontradas e Soluções Implementadas**

* **Alocação Dinâmica de Memória:**
  + *Dificuldade:* Gerenciar a alocação dinâmica de memória para o tabuleiro.
  + *Solução:* Utilização de ponteiros e funções de alocação de memória (**malloc**) para criar e liberar o tabuleiro.
* **Implementação da Lógica do Computador:**
  + *Dificuldade:* Criar uma estratégia simples para o computador fazer jogadas.
  + *Solução:* Estratégia básica de ocupar a primeira posição vazia encontrada.
* **Ranking e Manipulação de Arquivos:**
  + *Dificuldade:* Ler e escrever dados do ranking em um arquivo.
  + *Solução:* Utilização de funções **fopen**, **fprintf** e **fclose** para salvar e carregar o ranking.

**Demonstrativo das Funcionalidades Implementadas**

A seguir, são apresentados quatro prints de tela demonstrando a execução do programa:

* **Menu Principal:**
  + *Descrição:* Apresenta as opções do menu principal. 
* **Jogada do Usuário:**
  + *Descrição:* Exemplo de uma jogada válida do usuário (X). 
* **Jogada do Computador:**
  + *Descrição:* O computador (O) faz uma jogada automática. 
* **Resultado do Jogo:**
  + *Descrição:* Fim do jogo, mostrando o resultado (vitória, empate, etc.). 

APÊNDICE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Estrutura para armazenar dados do jogador

struct Jogador {

char nome[50];

int pontuacao;

};

// Protótipos de funções

void mostrarMenu();

void jogarJogo(int tamanhoTabuleiro);

void exibirTabuleiro(char \*\*tabuleiro, int tamanho);

int realizarJogada(char \*\*tabuleiro, int linha, int coluna, char jogador, int tamanho);

int verificarVitoria(char \*\*tabuleiro, char jogador, int tamanho);

int verificarEmpate(char \*\*tabuleiro, int tamanho);

void jogadaComputador(char \*\*tabuleiro, char jogador, int tamanho);

void mostrarRanking(struct Jogador \*jogadores, int numJogadores);

void salvarRanking(struct Jogador \*jogadores, int numJogadores);

int main() {

int opcao;

do {

mostrarMenu();

scanf("%d", &opcao);

switch (opcao) {

case 1:

jogarJogo(3); // Pode ser alterado para 5 para um tabuleiro 5x5

break;

case 2:

printf("Ranking:\n");

// Carregar dados do arquivo ou inicializar uma estrutura de jogadores

struct Jogador jogadores[10]; // Supondo um máximo de 10 jogadores

mostrarRanking(jogadores, 10);

break;

case 3:

printf("Créditos: Desenvolvido por [seu nome]\n");

break;

case 4:

printf("Saindo do jogo. Até logo!\n");

break;

default:

printf("Opção inválida. Tente novamente.\n");

}

} while (opcao != 4);

return 0;

}

void mostrarMenu() {

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

printf("1. Jogar\n");

printf("2. Ver Ranking\n");

printf("3. Créditos\n");

printf("4. Sair\n");

printf("Escolha a opção: ");

}

void jogarJogo(int tamanhoTabuleiro) {

char \*\*tabuleiro;

char jogadorAtual = 'X';

int linha, coluna, jogadaValida;

// Alocação dinâmica do tabuleiro

tabuleiro = (char \*\*)malloc(tamanhoTabuleiro \* sizeof(char \*));

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

tabuleiro[i] = (char \*)malloc(tamanhoTabuleiro \* sizeof(char));

memset(tabuleiro[i], ' ', tamanhoTabuleiro);

}

do {

exibirTabuleiro(tabuleiro, tamanhoTabuleiro);

if (jogadorAtual == 'X') {

printf("Jogador X, insira a linha (0-%d) e coluna (0-%d) separadas por espaço: ", tamanhoTabuleiro - 1,

tamanhoTabuleiro - 1);

scanf("%d %d", &linha, &coluna);

jogadaValida = realizarJogada(tabuleiro, linha, coluna, jogadorAtual, tamanhoTabuleiro);

} else {

printf("Computador (Jogador O) está pensando...\n");

jogadaComputador(tabuleiro, jogadorAtual, tamanhoTabuleiro);

jogadaValida = 1; // A jogada do computador é sempre válida

}

if (jogadaValida) {

if (verificarVitoria(tabuleiro, jogadorAtual, tamanhoTabuleiro)) {

exibirTabuleiro(tabuleiro, tamanhoTabuleiro);

if (jogadorAtual == 'X') {

printf("Jogador X venceu!\n");

} else {

printf("Computador (Jogador O) venceu!\n");

}

break;

}

if (verificarEmpate(tabuleiro, tamanhoTabuleiro)) {

exibirTabuleiro(tabuleiro, tamanhoTabuleiro);

printf("O jogo terminou em empate!\n");

break;

}

jogadorAtual = (jogadorAtual == 'X') ? 'O' : 'X';

} else {

printf("Jogada inválida. Tente novamente.\n");

}

} while (1);

// Liberação de memória

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

free(tabuleiro[i]);

}

free(tabuleiro);

}

void exibirTabuleiro(char \*\*tabuleiro, int tamanho) {

printf("\n");

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

for (int j = 0; j < tamanho; ++j) {

printf("%c ", tabuleiro[i][j]);

}

printf("\n");

}

}

int realizarJogada(char \*\*tabuleiro, int linha, int coluna, char jogador, int tamanho) {

if (linha >= 0 && linha < tamanho && coluna >= 0 && coluna < tamanho && tabuleiro[linha][coluna] == ' ') {

tabuleiro[linha][coluna] = jogador;

return 1;

} else {

return 0;

}

}

int verificarVitoria(char \*\*tabuleiro, char jogador, int tamanho) {

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

// Verifica linhas e colunas

if ((tabuleiro[i][0] == jogador && tabuleiro[i][1] == jogador && tabuleiro[i][2] == jogador) ||

(tabuleiro[0][i] == jogador && tabuleiro[1][i] == jogador && tabuleiro[2][i] == jogador)) {

return 1;

}

}

// 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;

}

return 0;

}

int verificarEmpate(char \*\*tabuleiro, int tamanho) {

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

for (int j = 0; j < tamanho; ++j) {

if (tabuleiro[i][j] == ' ') {

return 0;

}

}

}

return 1;

}

void jogadaComputador(char \*\*tabuleiro, char jogador, int tamanho) {

// Estratégia simples: joga na primeira posição vazia encontrada

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

for (int j = 0; j < tamanho; ++j) {

if (tabuleiro[i][j] == ' ') {

tabuleiro[i][j] = jogador;

return;

}

}

}

}

void mostrarRanking(struct Jogador \*jogadores, int numJogadores) {

// Carrega dados do arquivo ou inicializa jogadores com pontuações fictícias

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

strcpy(jogadores[i].nome, "Jogador");

jogadores[i].pontuacao = rand() % 100; // Pontuação fictícia

}

// Classifica os jogadores pelo ranking (pontuação)

for (int i = 0; i < numJogadores - 1; ++i) {

for (int j = i + 1; j < numJogadores; ++j) {

if (jogadores[i].pontuacao < jogadores[j].pontuacao) {

// Troca de posições

struct Jogador temp = jogadores[i];

jogadores[i] = jogadores[j];

jogadores[j] = temp;

}

}

}

// Exibe o ranking

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

printf("%d. %s - Pontuação: %d\n", i + 1, jogadores[i].nome, jogadores[i].pontuacao);

}

// Salva o ranking no arquivo

salvarRanking(jogadores, numJogadores);

}

void salvarRanking(struct Jogador \*jogadores, int numJogadores) {

FILE \*arquivo = fopen("ranking.txt", "w");

if (arquivo == NULL) {

printf("Erro ao abrir o arquivo de ranking.\n");

return;

}

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

fprintf(arquivo, "%s %d\n", jogadores[i].nome, jogadores[i].pontuacao);

}

fclose(arquivo);

}