ORDEM:

-Importações e Bibliotecas
//Esse trecho do código é responsável por importar bibliotecas essenciais que permitem que o jogo da velha online funcione corretamente, tanto no visual (cores e tela) quanto na comunicação em rede (modo online).

-Criação Do Tabuleiro
//Esse trecho do código é responsável por controlar a lógica visual e estrutural do tabuleiro do jogo da velha. Vamos entender cada parte

-Núcleo Do Jogo
//Ela trata da lógica principal de jogabilidade entre dois jogadores conectados via rede (um como servidor e outro como cliente).

-Modo Servido
// Essa função é responsável por inicializar o programa como servidor TCP, esperando a conexão do cliente para iniciar o jogo da velha

-Modo Cliente
//Ela é a parte do cliente da conexão TCP, ou seja, quem se conecta ao servidor para jogar como o jogador 2 (O).

-Menu Do Jogo
//Essa é o ponto de entrada do programa. Quando você executa o jogo da velha online, é aqui que tudo começa.

Importações e Bibliotecas. Linha 1-9

#include <stdio.h> //Usada para entrada e saída padrão. Fornece funções como printf() e scanf().

#include <stdlib.h> // Contém funções para alocação de memória, controle de processos e //conversão, como exit(), system() etc.

#include <string.h> // Fornece funções de manipulação de strings, como strcmp(), strcpy() etc.

#include <winsock2.h> // Responsável por fornecer acesso à API de sockets no Windows

#include <windows.h> // Biblioteca do sistema operacional Windows que dá acesso a funções //gerenciamento de janelas, processos, entre outros.

Vinculação de Bibliotecas.

#pragma comment(lib, "ws2_32.lib") // Isso linka automaticamente o programa com a biblioteca //ws2_32.lib, que é a implementação da Winsock 2

Definição de Porta.

#define PORTA 5555 // Cria um macro, substituindo todas as ocorrências de PORTA no código por 5555.

Resumo:
Esse bloco inicial configura tudo que o programa vai precisar para:

Trabalhar com rede (Winsock)
Controlar o console (limpar tela, mudar cor)
Fazer entrada e saída básica
Definir a porta padrão da conexão TCP

Criação do Tabuleiro. Linha 11-92

```
char tabuleiro[9];
//Declara um vetor de 9 caracteres representando as 9 casas do tabuleiro.
//Inicialmente preenchido com '0' a '8' (feito pela função iniciarTabuleiro()).
//Durante o jogo, cada posição será preenchida com 'X' ou 'O'

void setCor(int cor)
//Altera a cor do texto no terminal do Windows.
//Usa funções da API do Windows:

GetStdHandle(STD_OUTPUT_HANDLE)
//obtém o terminal atual.

SetConsoleTextAttribute
//aplica a cor.


void limparTela()
//Essa função limpa a tela de forma segura e sem “piscar” como acontece com system("cls").
//Obtém o handle do terminal.
//Lê o tamanho da tela com GetConsoleScreenBufferInfo.
//Preenche todos os caracteres da tela com espaço (' ').
//Reseta os atributos de cor.
//Move o cursor para o canto superior esquerdo ((0,0)).
//Isso evita "tela preta piscando" e é mais elegante do que comandos externos.

void desenharTabuleiro()
//Essa função desenha o tabuleiro no terminal, com cores diferentes para cada símbolo:
//Limpa a tela antes de desenhar.
//Mostra cabeçalho: "TIC - TAC - TOE".
//Percorre as 9 casas:
//Se a casa contém 'X', pinta de vermelho.
//Se contém 'O', pinta de azul.
//Caso contrário (número), pinta de cinza claro.
//Exibe o símbolo e reseta a cor.
//Adiciona divisórias verticais | e horizontais --- conforme a posição.
//Ao fim, imprime a borda inferior do tabuleiro.
```

```
char verificarVencedor()
//Verifica se algum jogador venceu, avaliando as 8 possíveis combinações
de vitória:

int v[8][3] = {
    {0,1,2}, {3,4,5}, {6,7,8},  // Linhas
    {0,3,6}, {1,4,7}, {2,5,8},  // Colunas
    {0,4,8}, {2,4,6}            // Diagonais
};
//Para cada combinação:
//Se os três índices contêm o mesmo símbolo (X ou O), retorna esse
símbolo.
//Se ninguém ganhou, retorna ' '.

int empate()
//Detecta empate:
//Percorre todas as 9 posições do tabuleiro.
//Se encontrar alguma casa que não é 'X' nem 'O', retorna 0 (não é
empate).
//Se todas as casas estiverem preenchidas com 'X' ou 'O', retorna 1.

void iniciarTabuleiro()
//Preenche o vetor tabuleiro com os números de '0' a '8':

for (int i = 0; i < 9; i++)
    tabuleiro[i] = '0' + i;
//Importância:
//Serve tanto como identificador da casa no início do jogo...
//...quanto para saber se a casa ainda está disponível para jogada.

Resumo:
tabuleiro[9] // Armazena o estado do jogo
setCor() // Altera cor do texto no console
limparTela() // Limpa o terminal de forma segura
desenharTabuleiro() // Mostra o tabuleiro visual com cores
verificarVencedor() // Verifica se alguém ganhou
empate() // Verifica se o jogo terminou em empate
iniciarTabuleiro() // Inicializa o tabuleiro com valores de 0 a 8

-------

 Núcleo do jogo. Linha 95-161

 Assinatura da Função.

void jogar(SOCKET conexao, int jogadorLocal)
SOCKET conexao // representa a conexão de rede ativa com o outro jogador.

int jogadorLocal // define se o jogador atual é o jogador 1 (X) ou
jogador 2 (O).

 Inicialização.
```

```
int jogada;
char buffer[2];
iniciarTabuleiro();
jogada // guarda o índice (0-8) da casa escolhida no tabuleiro.

buffer[2] //  usado para enviar/receber a jogada pelo socket. Só 1
caractere é usado, o outro é sobra.

iniciarTabuleiro() //  zera o tabuleiro, preenchendo com os números '0' a
'8'

 Laço principal do jogo.

while (1) {
    desenharTabuleiro();
    ...
}
//Esse laço continua até o jogo acabar por vitória ou empate.

 Checagem de fim de jogo.

if (verificarVencedor() != ' ') break;
if (empate()) break;
Verifica se alguém ganhou (verificarVencedor) ou se deu empate. Se sim,
sai do while.

  Jogada do jogador local

if (jogadorLocal == 1) {
    ...
    tabuleiro[jogada] = 'X';
    buffer[0] = jogada + '0';
    send(conexao, buffer, 1, 0);
}
//Se você é o jogador 1, joga como X:
//Lê a jogada com scanf.
//Valida se é número entre 0-8 e se a casa está livre. Atualiza o
tabuleiro. Envia a jogada para o outro jogador via send.

 Jogada do adversário

else {
    printf("Esperando jogada do adversario...\n");
    recv(conexao, buffer, 1, 0);
    jogada = buffer[0] - '0';
    if (jogada >= 0 && jogada <= 8) tabuleiro[jogada] = 'X';
}

//Se você não é o jogador 1 (ou seja, é o jogador 2), então espera o X
jogar.
//Recebe um byte (índice de jogada) via recv. Atualiza o tabuleiro.

 Repetição para o jogador 2
```

```
//Depois da primeira parte, o processo se repete para o outro jogador:
//Se você é o jogador 2 (O), você joga agora.
//Se você é o jogador 1, espera o adversário jogar como O.

 Fim do jogo

desenharTabuleiro();
char vencedor = verificarVencedor();
...
//Exibe o tabuleiro final.
//Usa verificarVencedor() para saber quem ganhou.

//Mostra a mensagem:
Verde se venceu (setCor(10)),
Amarelo se empatou (setCor(14)),
Vermelho se perdeu (setCor(12)).

 Espera final

getchar(); getchar();
//Dois getchar()s para esperar o usuário apertar ENTER duas vezes antes
de encerrar.

Resumo:
A função jogar:
Controla o fluxo do jogo da velha online.
Alterna entre receber e enviar jogadas via rede.
Checa o estado do jogo a cada rodada.
Mostra quem venceu ou se houve empate.

-------

Criação da Partida/Entrando como Servidor. Linha 164-187

 Variáveis e estruturas

WSADATA wsa;
SOCKET s, cliente;
struct sockaddr_in servidor, remoto;
int tamanho = sizeof(remoto);
WSADATA wsa
// estrutura usada pelo Windows para inicializar a biblioteca Winsock.

SOCKET s // o socket principal do servidor (escuta)

SOCKET cliente // socket retornado após aceitar a conexão do cliente

sockaddr_in servidor // estrutura com as informações do servidor (IP,
porta)

sockaddr_in remoto // será preenchida com os dados do cliente que se
conectar

tamanho // necessário para a função accept
```

Inicialização da Winsock

```
WSAStartup(MAKEWORD(2, 2), &wsa);
//Inicia a biblioteca Winsock 2.2.
```

Criação do socket

```
s = socket(AF_INET, SOCK_STREAM, 0);
Cria um socket IPv4 (AF_INET) do tipo TCP (SOCK_STREAM).
//Retorna um identificador de socket (s).
```

Configuração do socket

```
servidor.sin_family = AF_INET;
servidor.sin_addr.s_addr = INADDR_ANY;
servidor.sin_port = htons(PORTA);
//Define o protocolo IPv4 (AF_INET).
```

INADDR_ANY → aceita conexões em qualquer IP da máquina (localhost, rede, etc.).

htons(PORTA) → converte a porta 5555 para formato de rede (big-endian).

Bind e listen

```
bind(s, (struct sockaddr *)&servidor, sizeof(servidor));
listen(s, 1);
```

bind → associa o socket s à porta e IP definidos.

listen(s, 1) → coloca o socket em modo de escuta, aceitando até 1 conexão pendente.

Aceita conexão do cliente

```
cliente = accept(s, (struct sockaddr *)&remoto, &tamanho);
//Espera uma conexão (bloqueia até alguém conectar).
```

Quando um cliente conecta:
Retorna um novo socket cliente, exclusivo para comunicação.
Preenche remoto com o IP e porta do cliente.

Inicia o jogo

```
jogar(cliente, 1);
//Chama a função jogar, passando o socket cliente.
//O 1 indica que o servidor será o jogador 1 (X).
```

Finalização

```
closesocket(s);
WSACleanup();
```

closesocket(s) → fecha o socket do servidor

WSACleanup() → encerra o uso da biblioteca Winsock

Resumo:
A função modoServidor():
Inicia a Winsock.
Cria e configura o socket servidor.
Escuta uma conexão.
Aceita o cliente.
Começa o jogo como jogador 1 (X).
Fecha tudo ao final.

Entrando como Cliente. Linha 191-214

Declarações

```
WSADATA wsa;
SOCKET s;
struct sockaddr_in servidor;
char ip[32];
//WSADATA wsa: estrutura para inicializar a Winsock.

SOCKET s // socket usado para se conectar ao servidor

sockaddr_in servidor // estrutura que representa o IP e porta do servido

char ip[32] // armazena o IP digitado pelo usuário
```

Entrada do IP

```
printf("Digite o IP do servidor: ");
scanf("%s", ip);
//Solicita que o usuário digite o IP do servidor (ex: 127.0.0.1 ou IP da
rede local).
```

Inicialização da Winsock e criação do socket

```
WSAStartup(MAKEWORD(2, 2), &wsa);
s = socket(AF_INET, SOCK_STREAM, 0);
//WSAStartup inicia a Winsock 2.2.
socket cria um socket TCP IPv4.
```

Configuração do destino (servidor)

```
servidor.sin_family = AF_INET;
servidor.sin_addr.s_addr = inet_addr(ip);
servidor.sin_port = htons(PORTA);
//sin_family usa IPv4.
//inet_addr(ip) converte o IP digitado (string) para um formato numérico
(in_addr_t).
//htons(PORTA) converte a porta 5555 para o formato correto da rede.
```

Conexão com o servidor

```
connect(s, (struct sockaddr *)&servidor, sizeof(servidor));
//Tenta se conectar ao servidor que está rodando no IP e porta definidos
//Essa função bloqueia até a conexão ser bem-sucedida ou falhar

 Sucesso e início do jogo

printf("Conectado ao servidor!\n");
jogar(s, 2);
//Informa que a conexão foi feita.
Inicia a função jogar, passando o socket s e indicando que este jogador é
o jogador 2 (O).

 Encerramento

closesocket(s);
WSACleanup();
//Fecha o socket do cliente.
//Encerra o uso da Winsock.

 Resumo:
A função modoCliente():
Solicita o IP do servidor.
Inicializa a Winsock.
Cria o socket.
Conecta ao servidor no IP/porta fornecidos.
Inicia o jogo como jogador 2 (O).
Finaliza recursos após o fim do jogo

-------

Menu do Jogo. Linha 217-231

 Limpa a tela

limparTela();
//Chama a função que limpa o console de forma segura.
//Isso garante que o menu seja exibido "limpinho", sem sujeira de
execução anterior.

 Exibe o menu

printf("=== JOGO DA VELHA ONLINE ===\n");
printf("1 - Criar partida (Servidor)\n");
printf("2 - Entrar na partida (Cliente)\n");
printf("Escolha: ");
//Mostra um menu simples com duas opções:
//Criar a partida (agir como servidor)
//Entrar numa partida (agir como cliente)

  Lê a escolha do usuário

scanf("%d", &opcao);
//Lê a opção digitada pelo usuário e armazena em opcao
```

```
 Executa a ação correspondente

if (opcao == 1) modoServidor();
else if (opcao == 2) modoCliente();
else printf("Opção inválida!\n");
//Se o usuário escolheu 1, inicia o modo servidor (espera o outro jogador
conectar).
//Se escolheu 2, entra como cliente (digita o IP e conecta ao servidor).
//Qualquer outro número exibe "Opção inválida!".

 Fim do programa

return 0;
//Finaliza o programa com código de retorno 0, indicando que terminou com
sucesso.

Resumo:
A função main():
É o menu inicial do jogo.
Deixa o usuário escolher entre ser servidor ou cliente.
Chama a função correta (modoServidor ou modoCliente).
Serve como um hub de entrada simples e direto.
```