

## Introdução

O projeto Jokenboom consiste em um jogo de guerra estratégica inspirado no Jokenpô clássico, implementado em C com uso de sockets POSIX no Linux. Ele simula uma batalha entre um cliente (jogador humano) e um servidor (computador), com cinco opções de ataque. Cada ataque possui vantagens e fraquezas, baseando-se em regras fixas. O servidor se conecta por IPv4 ou IPv6, conforme especificado na linha de comando, e realiza partidas até o cliente desejar parar, mantendo um histórico de vitórias e derrotas.

## Estrutura da Solução

A solução é composta por cinco arquivos principais:

- “cliente.c”: implementa o cliente TCP que se conecta ao servidor, envia a jogada e recebe o resultado.
- “server.c”: implementa o servidor TCP, que processa as jogadas, gera respostas, trata erros e gerencia a partida.
- “common.c e common.h”: contêm funções auxiliares para manipulação de endereços e mensagens, além da definição do protocolo de comunicação.
- Makefile: Importante para a execução de todos os arquivos do programa.

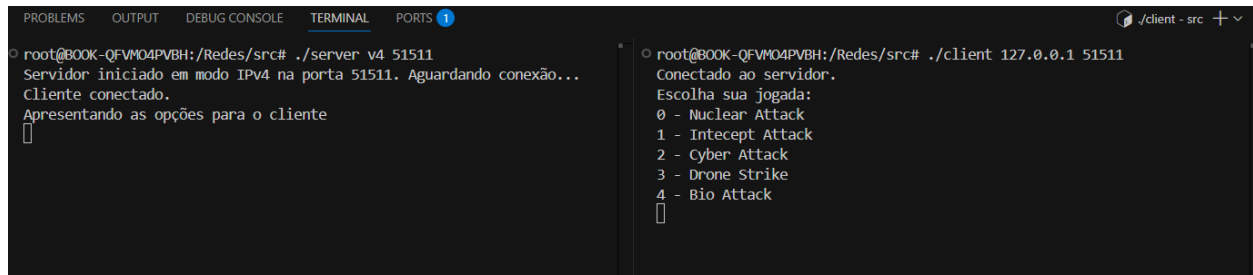
A comunicação ocorre via struct GameMessage, com tipos de mensagens bem definidos (MSG\_REQUEST, MSG\_RESPONSE, MSG\_RESULT, etc.) para garantir interoperabilidade.

## Funcionamento dos Códigos

O funcionamento dos códigos pode ser melhor descrito e visualizado por meio dos comentários escritos no código.

## Modo de execução e testes

Inicialmente, abra 2 terminais no vscode com tela dividida e execute o comando ‘make’ em algum deles. Após isso execute no primeiro terminal ‘./bin/server v4 51511’ (para IPv4) ou ‘./bin/server v6 51511’ (para IPV6). Isso permitirá a inicialização do servidor na porta 51511. No outro terminal execute ‘./bin/client 127.0.0.1 51511’. Isso permitirá a inicialização do cliente e a conexão dele com o servidor. Após essas ações teremos a seguinte tela mostrada na figura 1.

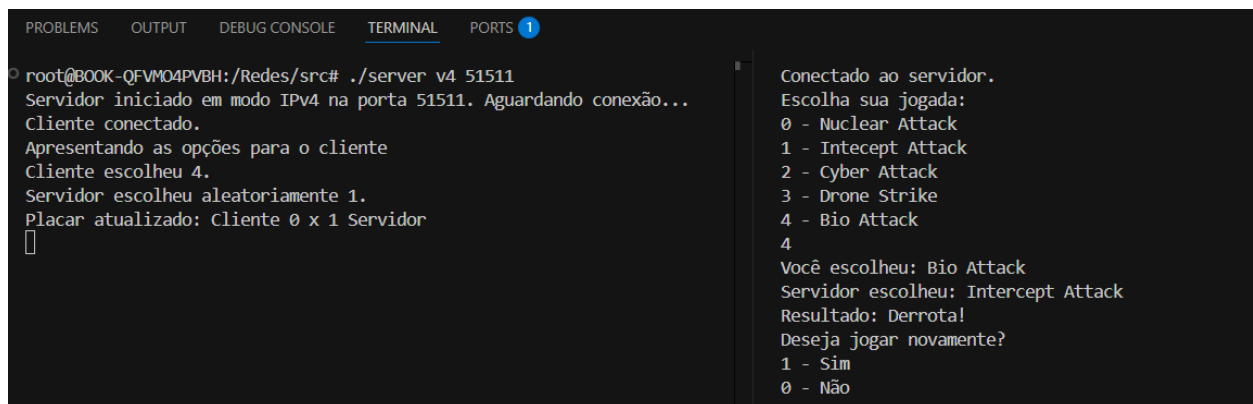


```
root@BOOK-QFVM04PVBH:/Redes/src# ./server v4 51511
Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
Cliente conectado.
Apresentando as opções para o cliente
[]

root@BOOK-QFVM04PVBH:/Redes/src# ./client 127.0.0.1 51511
Conectado ao servidor.
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
[]
```

**Figura 1:** Inicialização do servidor e do cliente

Em seguida será solicitado pelo servidor a escolha de uma jogada pelo cliente. Para demonstração, escolhendo a jogada 4 - Bio Attack, foi sorteado pelo servidor a jogada Intercept Attack e portanto o cliente perdeu, conforme mostrado na figura 2.

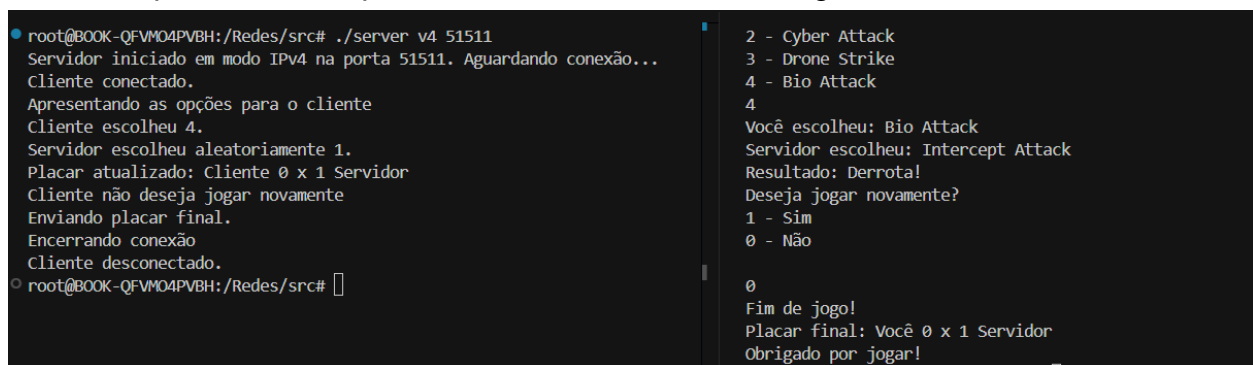


```
root@BOOK-QFVM04PVBH:/Redes/src# ./server v4 51511
Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
Cliente conectado.
Apresentando as opções para o cliente
Cliente escolheu 4.
Servidor escolheu aleatoriamente 1.
Placar atualizado: Cliente 0 x 1 Servidor
[]

Conectado ao servidor.
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
4
Você escolheu: Bio Attack
Servidor escolheu: Intercept Attack
Resultado: Derrota!
Deseja jogar novamente?
1 - Sim
0 - Não
```

**Figura 2:** Demonstração da jogada

Após isso é possível escolher entre continuar a jogar(1) ou parar de jogar(0). Para demonstração escolhi parar de jogar e com isso é encerrada a conexão e enviado pelo servidor o placar final da partida conforme mostrado na figura 3.



```
root@BOOK-QFVM04PVBH:/Redes/src# ./server v4 51511
Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
Cliente conectado.
Apresentando as opções para o cliente
Cliente escolheu 4.
Servidor escolheu aleatoriamente 1.
Placar atualizado: Cliente 0 x 1 Servidor
Cliente não deseja jogar novamente
Enviando placar final.
Encerrando conexão.
Cliente desconectado.
root@BOOK-QFVM04PVBH:/Redes/src# []

2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
4
Você escolheu: Bio Attack
Servidor escolheu: Intercept Attack
Resultado: Derrota!
Deseja jogar novamente?
1 - Sim
0 - Não

0
Fim de jogo!
Placar final: Você 0 x 1 Servidor
Obrigado por jogar!
[]
```

**Figura 3:** Demonstração do término da partida

## Testes de erro e validações

Também é possível testar as validações de número do teclado. No exemplo da figura 4 testei o digitar o valor 6 na escolha da jogada e com isso foi gerado um texto de erro pelo servidor e solicitado uma nova jogada.

```
root@BOOK-QFVM04PVBH:/Redes/src# ./server v4 51511
Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
Cliente conectado.
Apresentando as opções para o cliente
Cliente escolheu 6.
Erro: Opção inválida de jogada.
Apresentando as opções para o cliente

```

```
Conectado ao servidor.
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
6
Por favor, selecione um valor de 0 a 4.
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack

```

**Figura 4:** Teste de erro - jogada inválida

O mesmo ocorre para a validação se quer jogar novamente, conforme pode ser visto na figura 5.

```
root@BOOK-QFVM04PVBH:/Redes/src# ./server v4 51511
Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
Cliente conectado.
Apresentando as opções para o cliente
Cliente escolheu 6.
Erro: Opção inválida de jogada.
Apresentando as opções para o cliente
Cliente escolheu 2.
Servidor escolheu aleatoriamente 3.
Placar atualizado: Cliente 1 x 0 Servidor
Perguntando se o cliente deseja jogar novamente.
Erro: resposta inválida para jogar novamente.

```

```
4 - Bio Attack
2
Você escolheu: Cyber Attack
Servidor escolheu: Drone Strike
Resultado: Vitória!
Deseja jogar novamente?
1 - Sim
0 - Não
5
Por favor, digite 1 para jogar novamente ou 0 para encerrar.
Deseja jogar novamente?
1 - Sim
0 - Não

```

**Figura 5:** Teste de erro - Jogar novamente invalido

## Problemas encontrados e soluções:

Durante a realização do código enfrentei alguns desafios, tais como:

- Problema para configurar o linux no meu computador windows, pois não estava conseguindo executar o programa mesmo após a instalação do WSL Ubuntu. Com isso, realizei a instalação de uma outra versão do Ubuntu e instalei extensões no VsCode para suporte da execução.
- No início, tive bastante dificuldade para entender como o código conseguia tratar tanto conexões IPv4 quanto IPv6 de forma unificada. A mistura de estruturas diferentes (sockaddr\_in para IPv4 e sockaddr\_in6 para IPv6), o uso de sock\_add storage como tipo genérico e os diversos casts entre elas tornaram o entendimento um pouco confuso. Além disso, funções como inet\_pton e memcpy

aplicadas em contextos diferentes exigiram que eu revisasse como funciona a manipulação de endereços binários em C, especialmente para o caso do IPv6, que tem campos maiores e exige mais cuidado com alinhamento de memória. Porém, após analisar cada parte com calma, percebi que tudo se baseava em uma estrutura bem planejada e reutilizável. Percebi que o uso de `sockaddr_storage` foi a chave para permitir que o mesmo código funcionasse com os dois protocolos, já que permite aumentar o tamanho do armazenamento.

- No início do desenvolvimento, tive certa dificuldade para entender claramente quais partes da lógica deveriam estar no código do cliente e quais pertenciam ao servidor. Por falta de clareza inicial, acabei implementando algumas partes da lógica, como validações e decisões do jogo diretamente no cliente. No entanto, após reler com mais atenção o documento disponibilizado, percebi que a responsabilidade da lógica do jogo era inteiramente do servidor. O papel do cliente era apenas interagir com o usuário, ou seja, ler a ação digitada pelo jogador e exibir as mensagens enviadas pelo servidor. Depois que compreendi corretamente essa separação de responsabilidades realizei a reestruturação dos códigos
- Tive problemas ao mostrar na tela as escolhas do cliente e do servidor. Isso aconteceu porque esqueci de colocar o `\n` (quebra de linha) no final do `printf`. Sem esse caractere, a mensagem não era exibida imediatamente, já que o texto ficava "preso" em um buffer interno do programa. Ou seja, a mensagem só aparecia quando o buffer era esvaziado, o que dava a impressão de que o programa não estava funcionando corretamente. Foi só colocar o `\n` no final que tudo voltou a funcionar como esperado.
- Enfrentei dificuldades para implementar a lógica que pede um novo dígito (0 ou 1) ao cliente, no momento em que ele insere um valor fora do intervalo ao decidir se quer jogar novamente. O problema acontecia porque eu inicialmente acreditava que essa validação deveria ser feita no código do cliente, já que é para ele que o servidor envia a mensagem de erro(`MSG_ERROR`). A lógica implementada funcionava bem na primeira etapa de validação, aquela que verifica se a jogada está dentro do intervalo válido, pois nesse caso bastava apenas exibir a mensagem de erro. Porém, na validação de continuar ou não o jogo, também era necessário ler novamente a entrada do usuário via teclado, o que não estava sendo feito corretamente, mesmo fazendo diversas modificações nessa validação. A solução foi implementar essa lógica diretamente no código do servidor que ao receber um valor inválido, entra em um laço `while` que solicita ao cliente um novo valor até que um número válido (0 ou 1) seja inserido.