Trabalho prático 1 - Jokenboom

Entrega individual

Data de Entrega: 23 de Maio de 2025, às 23:59

1. Introdução

Imagine um futuro sombrio, consumido pelo caos da Terceira Guerra Mundial. As nações colapsaram em alianças instáveis, a diplomacia ruiu diante da escalada tecnológica do armamento bélico, e o planeta inteiro se transformou em um tabuleiro imprevisível de destruição. Em meio a esse cenário apocalíptico, você ocupa uma das posições mais críticas do planeta: chefe do setor de inteligência da OTAN. Sua missão é simples apenas em aparência — tomar uma única decisão militar. No entanto, essa escolha ocorre sob o mais absoluto silêncio estratégico: nenhum radar, nenhum relatório, nenhum espião. Nenhuma informação sequer sobre o que o inimigo está prestes a fazer.

Diante de você, cinco opções, cada uma com seu próprio potencial devastador: Ataque Nuclear, Interceptação de Mísseis, Ciberataque, Bombardeio com Drones e Emprego de Armas Biológicas. Cada ação representa um ramo da guerra moderna, com forças e fraquezas próprias, podendo ser decisiva ou fatal, dependendo da escolha simultânea feita pelo inimigo. O que poderia ser um embate de dados e previsões torna-se um duelo de instinto e risco, onde a lógica de combate segue uma estrutura inspirada no clássico **Rock**-**Paper**-**Scissors**-**Lizard**-**Spock**, de Sam Kass, agora adaptada para o palco mais letal da humanidade.

Aqui, a guerra não perdoa hesitações. Você escolhe, o inimigo escolhe, e o destino é selado em frações de segundo. Um ciclo tenso e imprevisível que se repete até que uma das partes saia vitoriosa — ou seja reduzida a cinzas. Neste jogo, a única constante são as consequências de longo alcance que cada decisão carrega. Este é o Jokenboom: um campo de batalha onde estratégia e sorte colidem, e onde cada escolha pode ser a última.

2. Objetivos

2.1. Objetivo Geral

Desenvolver um sistema cliente-servidor para a realização de partidas de Jokenpo expandido com 5 opções entre um jogador humano (cliente) e o computador (servidor). A dinâmica consiste em rodadas consecutivas de jogo, onde o cliente escolhe manualmente uma das opções disponíveis, enquanto o servidor realiza uma escolha aleatória. Em caso de empate, novas rodadas são realizadas até que se

determine um vencedor. Após cada partida concluída, o cliente tem a opção de jogar novamente ou encerrar a sessão. O sistema deve permitir ao servidor operar tanto em redes IPv4 quanto IPv6, conforme especificado por parâmetro, e manter um histórico de vitórias do cliente e do servidor durante a sessão.

2.2. Regras

- **Processamento das jogadas:** Determinar o vencedor de cada confronto utilizando a seguinte tabela de regras:

Ação	Vence de…	Perde para…
Nuclear Attack	Cyber Attack, Drone Strike	Intercept Attack, Bio Attack
Intercept Attack	Nuclear Attack, Bio Attack	Cyber Attack, Drone Strike
Cyber Attack	Intercept Attack, Drone Strike	Nuclear Attack, Bio Attack
Drone Strike	Intercept Attack, Bio Attack	Nuclear Attack, Cyber Attack
Bio Attack	Nuclear Attack, Cyber Attack	Intercept Attack, Drone Strike

3. Requisitos

3.1. Requisitos Funcionais

3.1.1 Conexão:

- O servidor deve ser inicializado com dois parâmetros:
  - Protocolo: **v4** para IPv4 ou **v6** para IPv6
  - Porta: para definir a porta de escuta

- O cliente conecta-se via TCP ao servidor com dois parâmetros:
  - Endereço: ip do servidor.
  - Porta: definida pelo servidor.

3.1.2 Realização do Duelo:

- O servidor envia uma mensagem solicitando que o cliente escolha sua ação dentre as seguintes opções:

Opção	Ação
0	Nuclear Attack
1	Intercept Attack
2	Cyber Attack
3	Drone Strike
4	Bio Attack

- O cliente envia sua escolha, que deve estar no intervalo de 0 a 4.
- O servidor gera uma escolha aleatória entre 0 e 4.

3.1.3 Determinação do Resultado:

- O servidor compara as escolhas de acordo com a tabela de regras definida e armazena o valor numérico do resultado.

Resultado	Descrição
1	Vitória
0	Derrota
-1	Empate

- Caso ocorra empate, o servidor requisita nova escolha até que haja um resultado de Vitória ou Derrota.

3.1.4 Tratamento de Erros:

- Se o cliente enviar um valor fora do intervalo 0-4, o servidor deve enviar uma mensagem de erro informando que a escolha é inválida (verificar seção 7).
    - O servidor deve então reenviar a solicitação de escolha, permitindo que o cliente faça uma nova jogada.

- Se o cliente enviar um valor diferente de 0 ou 1 para jogar novamente, o servidor deve enviar uma mensagem de erro informando que a escolha é inválida (verificar seção 7).
    - O servidor deve então reenviar a solicitação de escolha, permitindo que o cliente decida jogar mais uma vez ou não.

3.1.5 Feedback:

- Após a apuração do resultado, o servidor envia ao cliente as informações sobre:
    - A escolha do cliente
    - A escolha do servidor
    - O resultado (vitória do cliente ou vitória do servidor ou empate)

3.1.6 Jogar novamente

- Após um resultado decidido, sem empate, o servidor envia uma mensagem perguntando se o cliente deseja jogar novamente.
    - O cliente deve responder com '1' para jogar novamente ou '0' para encerrar a sessão.
        1. Se o cliente escolher jogar novamente, o servidor reinicia o processo de jogo.
        2. Se o cliente escolher encerrar, o servidor envia o placar final da sessão e encerra a conexão.

3.1.7 Encerramento

- O servidor encerra a conexão apenas quando o cliente opta por não jogar mais partidas.
- Antes do encerramento, o servidor envia uma mensagem final exibindo o placar completo da sessão.

3.2. Requisitos Não Funcionais

3.2.1 Linguagem: O projeto deve ser implementado em C, utilizando sockets POSIX.

3.2.2 Sistema Operacional: Linux.

3.2.3 Interface: Mensagens exibidas em terminal, sem GUI.

4. Comandos

4.1. Executando o servidor

Por exemplo, para inicializar o servidor utilizando o **IPv4** e usar a porta **51511**, execute:

```
./server v4 51511
```

O servidor exige dois argumentos:

- **<protocol>:** v4 (IPv4) | v6 (IPv6)
- **<port>:** A porta na qual o servidor está ouvindo.

4.2. Executando o Cliente

Por exemplo, para conectar ao servidor em **127.0.0.1** na porta **51511**, execute:

```
./client 127.0.0.1 51511
```

O cliente necessita de dois argumentos:

- **<server_ip>:** O IP (ou hostname) do servidor (por exemplo, 127.0.0.1 se estiver na mesma máquina).
- **<port>:** A porta na qual o servidor está ouvindo (deve ser igual à usada no servidor).

5. Arquitetura

5.1. Diagrama de Comunicação

A arquitetura deste trabalho é composta pela conexão entre um cliente e um servidor. Após o servidor ser ativado, o cliente estabelece a conexão. Em seguida, o servidor solicita ao cliente que escolha uma opção, o cliente envia sua escolha e o servidor retorna o resultado para o cliente. Após cada partida concluída, o servidor pergunta se o cliente deseja jogar novamente. Este ciclo se repete até que o cliente decida encerrar a sessão, momento em que o servidor envia o placar final e fecha a conexão.

6. Protocolo de Comunicação

6.1. Estrutura da Mensagem

A estrutura de mensagem para troca de informações via sockets entre cliente e servidor deve ser a seguinte:

```c
#define MSG_SIZE 256

typedef enum {
    MSG_REQUEST,
    MSG_RESPONSE,
    MSG_RESULT,
    MSG_PLAY_AGAIN_REQUEST,
    MSG_PLAY_AGAIN_RESPONSE,
    MSG_ERROR,
    MSG_END
} MessageType;
```

```c
typedef struct {
    int type;           // Tipo da mensagem
    int client_action;
    int server_action;
    int result;
    int client_wins;
    int server_wins;
    char message[MSG_SIZE];
} GameMessage;
```

Observação: é muito importante que a mensagem siga exatamente essa estrutura para que haja compatibilidade entre os clientes e servidores de diferentes alunos.

6.2. Mensagens

Tipo	Quem envia	Objetivo
MSG_REQUEST	Servidor	Para solicitar que o cliente envie uma jogada
MSG_RESPONSE	Cliente	Para enviar a jogada escolhida ao servidor
MSG_RESULT	Servidor	Para enviar o resultado após processar jogada
MSG_PLAY_AGAIN_REQUEST	Servidor	Para perguntar se o cliente deseja jogar novamente
MSG_PLAY_AGAIN_RESPONSE	Cliente	Para informar ao servidor se deseja jogar novamente
MSG_ERROR	Servidor	Quando a jogada do cliente for inválida
MSG_END	Servidor	Quando o jogo termina

6.3. Fluxo de Mensagens

6.3.1. Solicitação de Escolha:

- **Servidor → Jogador:** Envio de **MSG_REQUEST** solicitando que o cliente escolha uma ação.

6.3.2. Resposta do Cliente:

- **Cliente → Servidor:** O jogador responde com **MSG_RESPONSE**, informando a ação escolhida.

6.3.3. Processamento e Resultado:

- **Servidor:** Verifica se a escolha é válida (0-4).
    i. Se for inválida, envia **MSG_ERROR** com uma mensagem explicativa e retorna ao passo 1 (enviando **MSG_REQUEST** novamente).
    ii. Se for válida, compara com sua própria escolha aleatória e determina o resultado conforme a tabela de regras.

6.3.4. Envio do Resultado:

- **Servidor → Cliente:** Envio de **MSG_RESULT** contendo:
    i. A escolha do cliente (**client_action**)
    ii. A escolha do servidor (**server_action**)
    iii. O resultado do jogo (**result**)
    iv. Descrição textual do resultado (**message**)

6.3.5. Verificação de empate:

- Se houve empate: Retorno ao passo 6.3.1, repetindo o ciclo até que haja um vencedor.
- Se houve um vencedor: Prosseguir para o passo 6.3.6.

6.3.6. Pergunta sobre jogar novamente:

- **Servidor → Cliente:** Envio de **MSG_PLAY_AGAIN_REQUEST** perguntando se o cliente deseja jogar novamente.

6.3.7. Resposta sobre jogar novamente:

- **Cliente → Servidor:** O cliente responde com **MSG_PLAY_AGAIN_RESPONSE**.
    i. i. Se a resposta for 1 (sim), retorno ao passo 6.3.1 para iniciar uma nova partida.
    ii. ii. Se a resposta for 0 (não), prosseguir para o passo 6.3.8.

6.3.8. Encerramento da sessão:

- **Servidor → Cliente:** Envio de **MSG_END** contendo:
    i. O número de vitórias do cliente (**client_wins**)
    ii. O número de vitórias do servidor (**server_wins**)
    iii. Descrição textual do placar final (**message**)

7. Tratamento de erros

O usuário pode vir a digitar uma opção inválida. Esse erro deve ser tratado do lado do **servidor**. Os demais casos não mapeados não necessitam de uma mensagem de erro.

Erro	Mensagem
Escolha de jogada fora do intervalo 0–4	Por favor, selecione um valor de 0 a 4.
Escolha de jogar novamente fora do intervalo de 0–1	Por favor, digite 1 para jogar novamente ou 0 para encerrar.

8. Exemplos de uso

8.1. IPv4

Derrota e decisão de não jogar novamente

Terminal do Cliente	Terminal do Servidor
	$ ./server v4 51511 Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
$ ./client 127.0.0.1 51511 Conectado ao servidor.	Cliente conectado.

Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack 2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 2 Você escolheu: Cyber Attack Servidor escolheu: Nuclear Attack Resultado: Derrota!	Apresentando as opções para o cliente. Cliente escolheu 2. Servidor escolheu aleatoriamente 0. Placar atualizado: Cliente 0 x 1 Servidor
Deseja jogar novamente? 1 - Sim 0 - Não $ 0 Fim de jogo! Placar final: Você 0 x 1 Servidor Obrigado por jogar!	Perguntando se o cliente deseja jogar novamente. Cliente não deseja jogar novamente. Enviando placar final. Encerrando conexão. Cliente desconectado.

Empate seguido de Vitória e decisão de jogar novamente

Terminal do Cliente	**Terminal do Servidor**
	$ ./server v4 51511 Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
$ ./client 127.0.0.1 51511 Conectado ao servidor.	Cliente conectado.
Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack 2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 0 Você escolheu: Nuclear Attack Servidor escolheu: Nuclear Attack Resultado: Empate!	Apresentando as opções para o cliente. Cliente escolheu 0. Servidor escolheu aleatoriamente 0. Jogo empatado. Solicitando ao cliente mais uma escolha.
Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack	Apresentando as opções para o cliente. Cliente escolheu 2. Servidor escolheu aleatoriamente 3. Placar atualizado: Cliente 1 x 0 Servidor.

2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 2 Você escolheu: Cyber Attack Servidor escolheu: Drone Strike Resultado: Vitória!	
Deseja jogar novamente? 1 - Sim 0 - Não $ 1	Perguntando se o cliente deseja jogar novamente. Cliente deseja jogar novamente.
Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack 2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 3 Você escolheu: Drone Strike Servidor escolheu: Bio Attack Resultado: Vitória!	Apresentando as opções para o cliente. Cliente escolheu 3. Servidor escolheu aleatoriamente 4. Placar atualizado: Cliente 2 x 0 Servidor

Escolha de opções incorretas

Terminal do Cliente	**Terminal do Servidor**
	$ ./server v4 51511 Servidor iniciado em modo IPv4 na porta 51511. Aguardando conexão...
$ ./client 127.0.0.1 51511 Conectado ao servidor.	Cliente conectado.
Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack 2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 5 Por favor, selecione um valor de 0 a 4.	Apresentando as opções para o cliente. Cliente escolheu 5. Erro: opção inválida de jogada.

Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack 2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 2 Você escolheu: Cyber Attack Servidor escolheu: Drone Strike Resultado: Vitória!	Apresentando as opções para o cliente. Cliente escolheu 2. Servidor escolheu aleatoriamente 4. Fim de jogo. Encerrando conexão. Cliente desconectado. Encerrando servidor.
Deseja jogar novamente? 1 - Sim 0 - Não $ 2 Por favor, digite 1 para jogar novamente ou 0 para encerrar.	Perguntando se o cliente deseja jogar novamente. Erro: resposta inválida para jogar novamente.
Deseja jogar novamente? 1 - Sim 0 - Não $ 0 Fim de jogo! Placar final: Você 1 x 0 Servidor Obrigado por jogar!	Perguntando se o cliente deseja jogar novamente. Cliente não deseja jogar novamente. Enviando placar final. Encerrando conexão. Cliente desconectado.

8.2. IPv6

Vitória e decisão de não jogar novamente

Terminal do Cliente	**Terminal do Servidor**
	$ ./server v6 51511 Servidor iniciado em modo IPv6 na porta 51511. Aguardando conexão...
$ ./client ::1 51511 Conectado ao servidor.	Cliente conectado.
Escolha sua jogada: 0 - Nuclear Attack 1 - Intercept Attack	Apresentando as opções para o cliente. Cliente escolheu 1. Servidor escolheu aleatoriamente 0.

2 - Cyber Attack 3 - Drone Strike 4 - Bio Attack $ 1 Você escolheu: Intercept Attack Servidor escolheu: Nuclear Attack Resultado: Vitória!	
Deseja jogar novamente? 1 - Sim 0 - Não $ 0 Fim de jogo! Placar final: Você 1 x 0 Servidor Obrigado por jogar!	Perguntando novamente se o cliente deseja jogar novamente. Cliente não deseja jogar novamente. Enviando placar final. Encerrando conexão. Cliente desconectado.

9. Desenvolvimento

O projeto requer a implementação de dois componentes essenciais: um servidor e um cliente, ambos baseados no protocolo **TCP**. É crucial garantir que ambos os componentes sejam compatíveis **tanto com o IPv4 quanto com o IPv6**, proporcionando flexibilidade na escolha do endereço IP.

A entrada de mensagens do usuário diretamente do teclado é responsabilidade do cliente. O servidor, por sua vez, encarrega-se de todo o processamento do jogo.

É importante observar que, para este trabalho, **não é necessário que o servidor suporte múltiplos clientes simultaneamente.** Ele será projetado para lidar com conexões de um cliente por vez.

Para configurar o servidor corretamente, este deve ser capaz de receber, seguindo rigorosamente essa **ordem, o tipo de endereço desejado (v4 para IPv4 ou v6 para IPv6) e um número de porta** especificado na linha de comando. Recomenda-se a utilização da porta **51511** para manter a padronização do projeto.

Da mesma forma, o cliente precisa receber, também rigorosamente nessa ordem, o endereço IP do servidor e o número de porta para estabelecer a conexão com sucesso. Certifique-se de que ambos os componentes sejam configurados de acordo com essas diretrizes para uma integração eficaz e a comunicação adequada entre servidor e cliente.

Materiais para Consulta:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle.

- Playlist de programação com sockets do professor Ítalo Cunha

10. Avaliação

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C (não sendo permitida a utilização de C++)** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock.** **Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo (você pode testar com as implementações dos seus colegas)**. Procure escrever seu código de maneira clara, com comentários pontuais e bem identados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor.

Para a correção os seguintes testes serão realizados (com IPv4 e IPv6):

- Iniciar o servidor 10%
- Conexão do cliente estabelecida com o servidor 15%
- Funcionamento correto das jogadas do cliente para o servidor 10%
- Funcionamento correto da opção de jogar novamente 10%
- Contabilização correta de vitórias do cliente e servidor 10%
- Resultado correto do final do jogo 5%
- Fechamento de conexão 10%
- Tratamento de erros 5%
- Impressão correta das mensagens 5%
- Documentação 20%

11. Informações importantes

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte Arial tamanho 12, e figuras de tamanho adequado ao

tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.

Atenção: Será dado grande importância a parte da discussão dos desafios encontrados, pois as dificuldades ao longo do projeto sempre existem.

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.

Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O **Makefile** deve compilar o cliente em um binário chamado "**client**" e o servidor em um binário chamado "**server**" dentro de uma pasta chamada "**bin**" na raiz do projeto.
- **Seu programa deve ser compilado ao se executar apenas o comando "make", ou seja, sem a necessidade de parâmetros adicionais.**

- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP1_MATRICULA.zip

12. Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2d - 1$$

onde **d** representa o número de dias úteis de atraso.

Obs.: Note que após 3 dias, o trabalho não pode ser mais entregue.

13. Detecção de Trapaça

O trabalho prático desta disciplina **NÃO aceitará** trechos de código gerados por inteligência artificial, especificamente LLMs (Large Language Models). Para garantir a integridade acadêmica, utilizaremos um **analisador**, baseado no estado da arte, capaz de:

- Detectar com alta precisão padrões e “impressões digitais” de plataformas como **ChatGPT, DeepSeek, Gemini, Claude e afins**;

- Identificar tentativas de ofuscação ou modificações em código.

⚠️ **ATENÇÃO: NÃO HÁ “atalhos” possíveis**: qualquer sinal de código plagiado ou gerado resultará em nota **ZERO**. Sejam criativos, mas acima de tudo, **originais**.