Relatório Final: Redes

Alunos: **Isadora Venzke Dias - 22102745**
Victor Rafael - 22101443

Link para github: https://github.com/dvisa7178/BatalhaNaval.git

Protocolo: Batalha Naval

[1] Aplicação e Protocolo Desenvolvido

O objetivo deste projeto foi desenvolver um protocolo de camada de aplicação para um jogo de Batalha Naval multiusuário, seguindo a arquitetura cliente-servidor. A aplicação permite que dois jogadores se conectem a um servidor, que gerencia o estado da partida e a comunicação entre eles. Dessa forma, é possível executar o jogo que consiste em acertar alvos no tabuleiro (matriz) inimiga até que não reste nenhum ou antes que os seus sejam destruídos.

O protocolo define todas as interações necessárias para o funcionamento do jogo, incluindo a conexão dos jogadores, a sinalização de início de partida, a troca de mensagens de ataque, a notificação de resultados (acerto, erro ou navio afundado). Além disso, a alternância de turnos e a declaração de vitória ou derrota ao final do jogo também são gerenciados. O servidor atua como autoridade central, validando as jogadas e mantendo a sincronia do estado do jogo para ambos os clientes. É notável que as componentes, como protocolo, cliente e servidor estão separadas em vários arquivos, uma escolha para maior modularidade e debug durante a criação da aplicação.

[2] Características do Protocolo Desenvolvido

- **Cliente-Servidor:** Um servidor centralizado simplifica o gerenciamento do estado do jogo, a validação das regras e a sincronização das ações dos jogadores. Ele é o único responsável por determinar o resultado de um ataque e de quem é o próximo turno, evitando inconsistências. Dessa forma, 3 instâncias de terminal são necessárias: 2 clientes e um servidor.
- **Com Estado:** O servidor precisa manter o estado completo de cada partida em andamento, incluindo o tabuleiro de cada jogador, a posição de todos os navios e de quem é o turno atual. Sem estado, seria impossível gerenciar a lógica do jogo, uma vez que o resultado depende tanto de valores anteriores quanto pode demorar tempo imprevisível.
- **Persistente:** A conexão TCP entre cada cliente e o servidor é estabelecida no início e mantida aberta durante toda a partida. Isso reduz a latência, já que não é necessário reestabelecer a conexão a cada jogada.
- **Pull / Request-Response:** O cliente aguarda uma mensagem **ATAQUE_OPONENTE** do servidor (um "pull" de seu turno). Ao recebê-la, ele envia uma mensagem de ataque (**ATAQUE**). O servidor, então, processa o ataque e

responde com o resultado. Esse modelo é adequado por ser um jogo de turnos alternados.

- **Na Banda:** Todas as mensagens, tanto de dados (coordenadas de ataque) quanto de controle (início de turno, fim de jogo), trafegam pelo mesmo canal de comunicação (a mesma conexão TCP), simplificando a implementação.

[3] Tipos de Mensagens do Cliente para o Servidor

Tipo de Mensagem	Código	Descrição
INFO	300 INFO	Mensagem inicial enviada pelo cliente ao se conectar para estabelecer a comunicação e se identificar ao servidor. Serve como aviso e debug.
ATAQUE	100 ATAQUE	Envia as coordenadas (x,y) do ataque que o jogador deseja realizar durante o seu turno. Mensagem repetida a cada novo ataque.

[4] Tipos de Mensagens do Servidor para o Cliente

As mensagens do servidor são divididas em categorias de acordo com seus códigos. Pode-se observar os grupos abaixo:

Mensagens de Resposta/Status (Código 2xx):

Tipo de Mensagem	Código	Descrição
ESPERANDO_OPONENTE	201	Informa ao cliente que é o turno do oponente e que ele deve aguardar.
FIM_PARTIDA	202	Sinaliza o fim da partida para ambos os jogadores.

VENCEU	203	Notifica o jogador de que ele venceu a partida.
PERDEU	204	Notifica o jogador de que ele foi derrotado.
ATAQUE_OPONENTE	205	Informa ao jogador que é o seu turno de atacar.

Mensagens de Resultado de Ataque (Código 1xx):

Tipo de Mensagem	Código	Descrição
ATAQUE_FALHOU	100	O ataque do jogador errou o alvo (atingiu a água).
ATAQUE_ACERTO	101	O ataque do jogador acertou uma parte de um navio.
ATAQUES_SEQ	102	O jogador acertou alvos em sequência.
PORTA_AVIOES	103	Informa que um navio do tipo "Porta-aviões" foi afundado [tamanho 5].
ENCOURACADO	104	Informa que um navio do tipo "Encouraçado" foi afundado [tamanho 4].

CRUZADOR	105	Informa que um navio do tipo "Cruzador" foi afundado [tamanho 3].
SUBMARINO	106	Informa que um navio do tipo "Submarino" foi afundado [tamanho 3].
FRAGATA	107	Informa que um navio do tipo "Fragata" foi afundado [tamanho 2].
TORPEDEIRO	108	Informa que um navio do tipo "Torpedeiro" foi afundado [tamanho 1].

Mensagens de Informação (Código 3xx):

Tipo de Mensagem	Código	Descrição
INFO	300 INFO	Usada para enviar informações gerais, como o status de conexão e, principalmente, para transmitir o estado atualizado dos tabuleiros (pessoal e do inimigo) em formato JSON.

[5] Formato das Mensagens e Campos de Cabeçalho

O protocolo utiliza um formato de mensagem textual simples, composto por duas partes separadas por uma quebra de linha (\n):

- CÓDIGO\nCORPO.
- **Código:** Um campo de cabeçalho que identifica o tipo e a finalidade da mensagem.
- **Corpo:** O conteúdo da mensagem, cujo formato varia de acordo com o código.
- **Exemplos:**

1. **Ataque do Cliente:** O cliente envia o código 100 e, no corpo, as coordenadas separadas por vírgula. [Código: 100, Exemplo de ataque:3,5]
2. **Sinalização de Turno do Servidor:** O servidor envia o código 205 e uma mensagem textual informativa.

    a. Código: 205, Mensagem: Seu turno! Digite coordenadas x,y para atacar.
3. **Atualização de Mapa do Servidor:** O servidor envia o código 300 e um objeto JSON, no corpo da mensagem, contendo os mapas e uma mensagem de texto.
    a. 300
    b. {"text": "Acertou em (3,5) - Continue!", "own_map": [["~", ...]], "enemy_map": [["O", ...]]}

[6] Especificação dos Valores Possíveis dos Campos

Campo	Tipo	Valores Possíveis
CÓDIGO	String Numérica	**Cliente para Servidor:** 300, 100 **Servidor para Cliente:** 100, 101, 103-108, 201-205, 300
CORPO	String	Para 100 (Ataque): Coordenadas no formato "x,y" (ex: "4,7"). Para 300 (Info de Mapa): Uma string JSON serializada. Para os demais códigos: Uma string textual com descrição do evento (ex: "Acerto em (4,7)", "VOCÊ VENCEU!").

[7] Outras Informações Relevantes

- **Casos de Uso:**
    - O servidor é iniciado e aguarda por duas conexões.
    - O Jogador 1 executa cliente_ubuntu.py, insere o IP do servidor e se conecta. O servidor responde com uma mensagem de espera.
    - O Jogador 2 faz o mesmo. Ao se conectar, o servidor inicia a partida.
    - O servidor envia a mensagem **ATAQUE_OPONENTE**(205) para o Jogador 1 e **ESPERANDO_OPONENTE**(201) para o Jogador 2.
    - As posições dos navios são escolhidas de forma aleatória (simplificou debug e ajudou com alguns erros na hora de imprimir no terminal…).
    - O Jogador 1 envia um **ATAQUE**(100) com coordenadas.
    - O servidor processa o ataque, envia o resultado para ambos os jogadores e os mapas atualizados via mensagem **INFO**(300).
    - Se o Jogador 1 errar, o turno é alternado; se acertar, ele continua a jogar.
    - O ciclo se repete até que todos os navios de um jogador sejam afundados.
    - O servidor envia **VENCEU**(203) ao vencedor e **PERDEU**(204) ao perdedor, finalizando com **FIM_PARTIDA**(202).
- **APIs e Bibliotecas Utilizadas:**
    - **socket:** Para a comunicação TCP/IP de baixo nível.

    - **threading:** Utilizada no servidor para gerenciar as duas conexões de cliente simultaneamente sem bloquear a execução principal.
    - **json:** Para serializar e desseriliazar os dados dos tabuleiros, permitindo o envio de estruturas de dados complexas como texto.
- **Dificuldades Encontradas e Soluções:**
    - **Parsing de Mensagens em Buffer:** A recepção de dados via TCP pode agrupar múltiplas mensagens ou entregar mensagens parciais. Isso foi resolvido no cliente com a implementação de um message_buffer global, que armazena os dados recebidos e processa apenas mensagens completas (delimitadas por \n), garantindo que nenhuma mensagem seja perdida ou mal interpretada.
    - **Sincronização de Turnos:** Garantir que apenas um jogador pudesse atacar por vez foi crucial. A lógica de controle de estado no servidor, com as mensagens **ATAQUE_OPONENTE** e **ESPERANDO_OPONENTE**, resolveu essa questão de forma eficaz.
    - **Usabilidade e Diagnóstico de Rede:** Para facilitar o uso e a resolução de problemas de conexão (como firewall ou IPs incorretos), foram criados scripts auxiliares como diagnose_network.py e um menu inicial em start.sh.

[8] Desafios e Histórico

Considerando ser a primeira implementação real de redes feita, encontramos alguns desafios notáveis durante o desenvolvimento do trabalho. Dentre eles, destacam-se:

- **Imprimir no terminal:** Seja em nossa tentativa falha de usar cifra de Ceasar ou seja por quebrar o JSON, ficamos um certo tempo tentando imprimir a matriz de resultado só para obter: [Servidor]: {"text": "Jogo iniciado! Voc\u00ea \u00e9 o Player 1.", "own_map": [["", "", "", "", "", "", "", "", "", ""], ["", "", "B", "", "", "", "", "C", "", ""], ["", "", "B", "", "T", "", "", "C", "", ""], ["", "", "B", "", "", "", "", "C", "", ""], ["", "", "B", "", "", "", "", "", "", ""], ["", "", "", "", "", "A", "A", "A", "A", "A"], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["F", "F", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "S", "S", "S", ""]], "enemy_map": [["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""], ["", "", "", "", "", "", "", "", "", ""]]}300 INFO
- O problema acima parece ter sido ao usar “~” na água, mas não temos certeza. De uma forma ou outra, deixar uma matriz simples foi a solução!
- A comunicação entre máquina demandou certa atenção, uma vez que as versões iniciais do projeto funcionam perfeitamente com localhost, mas ao adicionar um segundo computador a mistura se perdia no IP e não conectava de forma alguma. Solucionado, após algum tempo.