Alunos: Leonardo Vieira Gonçalves Luciani Aquino Queiroz
Professor: Guilherme Corrêa
Disciplina: Redes de computadores
Data: 15 de ago. de 2025
Repositório: link_GitHub

Relatório do Protocolo - Jogo da Forca Multiplayer Cooperativo

Objetivo/Função da aplicação e protocolo desenvolvido

Este trabalho implementa um Jogo da Forca multiplayer cooperativo através de um protocolo de aplicação personalizado. A aplicação permite que múltiplos jogadores se conectem a um servidor central para colaborar na descoberta de palavras secretas em salas de jogo dedicadas. O protocolo desenvolvido tem como função principal gerenciar toda a comunicação entre clientes e servidor, padronizando as operações de: autenticação de jogadores, criação e administração de salas de jogo, sincronização do estado da partida entre clientes e servidor e controle de turnos.

O projeto foi desenvolvido em Java utilizando IntelliJ IDEA Community Edition. A seleção desta linguagem fundamentou-se em suas vantagens técnicas específicas: biblioteca de sockets de alto nível (java.net) que simplifica conexões cliente-servidor multiplataforma, suporte nativo robusto a threads para gerenciamento simultâneo de múltiplos clientes, e gerenciamento automático de memória via Garbage Collector, que elimina vulnerabilidades como vazamentos de memória e permite maior concentração na lógica de negócio. Adicionalmente, bibliotecas maduras como Gson facilitam significativamente a manipulação de dados JSON.

Características do protocolo desenvolvido

Arquitetura: Cliente-Servidor

- **Justificativa:** Escolhido para centralizar o controle do jogo em uma única autoridade, garantindo consistência do estado da partida e evitando conflitos de sincronização que poderiam ocorrer em arquiteturas peer-to-peer.

Modelo de Estado: Com Estado

- **Justificativa:** Necessário para manter informações persistentes sobre jogadores conectados, salas ativas, estado atual das partidas e histórico de tentativas, permitindo continuidade da experiência de jogo.

Persistência de Conexão: Persistente

- **Justificativa:** Conexões TCP mantidas abertas durante toda a sessão para permitir comunicação em tempo real, notificações instantâneas de eventos e reduzir latência nas jogadas.

Modelo de Comunicação: Híbrido (Push/Pull)

- **Justificativa:** O protocolo opera em modo **Pull** quando o cliente envia uma requisição e espera uma resposta direta (ex: NICK -> RESPOSTA_SERVIDOR). Opera em modo **Push** quando o servidor envia mensagens de forma proativa para os clientes, sem que eles as tenham solicitado naquele instante (ex: ATUALIZACAO_JOGO enviada a todos os jogadores após a jogada de um deles).

Controle: Na banda

- **Justificativa:** Mensagens de controle e dados trafegam pelo mesmo canal TCP, simplificando a implementação e reduzindo complexidade de gerenciamento de múltiplas conexões.

Protocolo de Transporte: TCP

- **Justificativa:** Garante entrega confiável e ordenada das mensagens, essencial para manter a consistência do estado do jogo.

Formato de Dados: JSON

- **Justificativa:** Escolhido pela legibilidade, facilidade de parsing e compatibilidade multiplataforma.

Tipos de mensagens do Cliente para o Servidor

Tipo de mensagem	Descrição
NICK	Define o apelido do jogador no servidor.
CRIAR	Solicita a criação de uma nova sala de jogo.
ENTRAR	Solicita a entrada em uma sala de jogo existente.
INICIAR_JOGO	Solicita o início da partida de forca na sala atual.
JOGAR_LETRA	Envia uma letra como palpite durante o turno do jogador.
SAIR	Informa ao servidor que o cliente está se desconectando.

Tipos de mensagens do Servidor para o Cliente

Tipo de mensagem	Descrição
RESPOSTA_SERVIDOR	Resposta direta a um comando do cliente (sucesso ou erro)
ATUALIZACAO_JOGO	Broadcast com o estado completo e atualizado do jogo para todos na sala
FIM_DE_JOGO	Broadcast informando o resultado final da partida (vitória/derrota).

Formato das mensagens e campos de cabeçalho

O formato geral para todas as mensagens é JSON, contendo um campo type e um payload. Com formato geral:

{

"type": "TIPO_DA_MENSAGEM",

"payload": { ... }

}

Exemplos Cliente -> Servidor:

NICK: {"type":"NICK", "payload":{"nickname":"jogador1"}}

CRIAR: {"type":"CRIAR", "payload":{"nomeSala":"herois", "capacidade":4}}

ENTRAR: {"type":"ENTRAR", "payload":{"nomeSala":"herois"}}

INICIAR_JOGO: {"type":"INICIAR_JOGO", "payload":{}}

JOGAR_LETRA: {"type":"JOGAR_LETRA", "payload":{"letra":"A"}}

SAIR: {"type":"SAIR", "payload":{}}

Exemplos Servidor -> Cliente:

RESPOSTA_SERVIDOR (Sucesso): {"type":"RESPOSTA_SERVIDOR", "payload":{"sucesso":true, "mensagem":"OK, sala 'herois' criada."}}

RESPOSTA_SERVIDOR (Erro): {"type":"RESPOSTA_SERVIDOR", "payload":{"sucesso":false, "mensagem":"ERRO: Sala está cheia."}}

ATUALIZACAO_JOGO: {"type":"ATUALIZACAO_JOGO", "payload":{"palavra":"_ A _ A _", "letrasTentadas":["E", "A"], "tentativasRestantes":5, "turnoDe":"jogador2"}}

FIM_DE_JOGO: {"type":"FIM_DE_JOGO", "payload":{"resultado":"VITORIA", "mensagem":"Parabéns!", "palavraCorreta":"JAVA"}}

Especificação dos valores dos campos e bibliotecas

Campos Principais:

- **type**: String identificando o tipo da mensagem
- **payload**: Objeto JSON contendo dados específicos da mensagem

Bibliotecas usadas:

- **java.net.*** - Comunicação de rede (ServerSocket, Socket)
- **java.io.*** - Entrada/saída de dados (BufferedReader, PrintWriter, InputStreamReader, IOException)
- **java.time.*** - Manipulação de data/hora (LocalDateTime, DateTimeFormatter)
- **java.util.*** - Estruturas de dados (ArrayList, HashMap, LinkedHashSet, List, Map, Set, Scanner)
- **java.util.concurrent.*** - Estruturas thread-safe (ConcurrentHashMap)
- **com.google.gson.*** - Serialização JSON (Gson, JsonSyntaxException)

Casos de uso principais

- **Criação de Sessão:** Cliente/Jogador define nickname → cria/entra em sala → inicia o jogo.
- **Sincronização:** Cliente e servidor recebem atualizações em tempo real.