Relatório T2

1. Introdução

Este trabalho tem como proposta adaptar um jogo single player desenvolvido em Go para um ambiente multiplayer utilizando RPC (Remote Procedure Call). A ideia principal é permitir que vários jogadores se conectem a um servidor central, compartilhem suas ações e posições e mantenham o mesmo estado de jogo.
A comunicação segue o modelo em que o cliente inicia as trocas de informação, consultando o servidor periodicamente para buscar atualizações do estado global.

2. Arquitetura do Sistema

Servidor

O servidor atua como o núcleo do sistema, responsável por armazenar e gerenciar o estado atual do jogo, incluindo a lista de jogadores e suas posições.
É importante destacar que o servidor não contém o mapa nem executa nenhuma lógica de movimentação ou colisão. Essas partes permanecem no cliente.
Além disso, todas as chamadas RPC e respostas são registradas no terminal para auxiliar na depuração e acompanhamento do funcionamento do sistema.

Cliente

O cliente concentra toda a interface gráfica e a lógica de jogo. É ele quem trata a movimentação do personagem, as interações e a atualização visual.
Ele também se comunica periodicamente com o servidor, tanto para enviar comandos quanto para receber o estado atualizado dos demais jogadores.
Cada cliente executa uma goroutine dedicada para buscar essas atualizações de forma assíncrona, o que garante fluidez e evita travamentos durante o jogo.

3. Comunicação e Consistência

Tratamento de Erros

As chamadas RPC possuem tratamento de falhas, permitindo que o cliente tente novamente em caso de erro de comunicação, garantindo mais estabilidade na conexão.

Execução Única (Exactly Once)

Cada comando que modifica o estado do jogo é identificado por um sequenceNumber.
O servidor armazena o último número processado de cada cliente e ignora repetições, evitando que a mesma ação seja executada mais de uma vez.

Concorrência

Tanto no cliente quanto no servidor são utilizados mutexes para proteger o acesso às variáveis compartilhadas.
Isso garante que múltiplas goroutines possam operar ao mesmo tempo sem causar inconsistências no estado do jogo.

4. Conclusão

A implementação final mantém uma separação clara entre servidor e cliente.
O servidor gerencia o estado global e garante a consistência entre os jogadores, enquanto o cliente cuida da lógica e da renderização.
Durante os testes, foi possível conectar múltiplos jogadores simultaneamente e observar que o estado do jogo permaneceu sincronizado entre todos.
O sistema mostrou-se estável, coerente com os requisitos da especificação e atende plenamente aos objetivos propostos para o trabalho.

5. Compilar

1. Abra o projeto na sua IDE e vá até o arquivo server.go.
Compile e execute com os comandos:

go build -o server.exe

./server

2. Em outro terminal, no mesmo diretório do projeto, compile e execute o cliente:

go build -o jogo.exe

./jogo <nome_do_jogador>

3. Quando for solicitado o endereço do servidor, digite:

127.0.0.1:8932

4. Para adicionar outro cliente, repita o mesmo processo em um novo terminal.