Relatório - Trabalho 2 - FPPD

Guilherme Barreto Goulart
Pedro Augusto Pereira
Rodrigo Oliveira Rosa

Escola Politécnica— PUCRS

19 de junho de 2024

Resumo

No presente artigo, apresentamos uma análise detalhada de um problema computacional que consiste em executar um programa cliente-servidor com múltiplas instâncias de clientes compartilhando remotamente os mesmos dados e interagindo com o servidor utilizando RPC, com o objetivo de demonstrar o funcionamento da computação distribuida.

Introdução

Este documento descreve o processo de desenvolvimento de um jogo multiplayer utilizando a linguagem Go e o protocolo RPC (Remote Procedure Call). O objetivo foi modificar um jogo originalmente de jogador único, transformando-o em um ambiente multiplayer onde múltiplos jogadores podem se conectar a um servidor central e jogar no mesmo mapa. A comunicação é iniciada pelos clientes, que periodicamente buscam atualizações do estado do jogo do servidor, garantindo uma experiência de jogo sincronizada.

Primeira solução

A solução foi desenvolvida a partir de uma versão de código fornecida pelo professor, seguindo a especificação de modificar um jogo de jogador único para um ambiente multiplayer. A arquitetura do sistema foi dividida em duas partes principais: o servidor de jogo (GameServer) e o cliente do jogo (GameClient).

Servidor de Jogo (GameServer)

- Gerenciamento do Estado do Jogo: O servidor é responsável por manter a matriz do mapa do jogo e as posições de todos os elementos dinâmicos, como jogadores, inimigos e itens.
- Comunicação com Clientes: Implementamos uma interface RPC que define métodos para registrar clientes, receber comandos de movimento/interação e fornecer o estado atual do jogo aos clientes.
- Processamento de Comandos: O servidor processa os comandos recebidos dos clientes, atualizando o estado do jogo conforme necessário.

Cliente do Jogo (GameClient)

- Interface Gráfica: Cada cliente possui uma interface gráfica onde o jogador interage com o jogo.
- Conexão com o Servidor: O cliente se conecta ao servidor para obter o estado do jogo e envia comandos de movimento e interação.
- Thread de Atualização: Cada cliente possui uma thread dedicada que periodicamente solicita atualizações do estado do jogo ao servidor e atualiza a interface gráfica do jogador.

Componentes e Interfaces

- GameServerInterface: Define métodos RPC como `RegisterClient`, `SendCommand`, e `GetGameState`.
- Estado do Jogo: Inclui a posição de todos os elementos dinâmicos e informações relevantes sobre os jogadores.

Execução e Resultados

Os resultados obtidos na implementação do programa desde a versão inicial(T1) refletem um progresso notável. Inicialmente, exploramos detalhadamente o problema proposto, discutindo possíveis abordagens e estratégias de implementação. Infelizmente, devido à escasses de tempo para refatorar nossa solução anterior para corrigir os erros e executar adequadamente, optamos por utilizar o código disponibilizado pelo professor, fazendo as alterações diretamente nele, sem considerar o código implementado pelo grupo anteriormente. No resumo, temos:

- Servidor: Iniciado primeiro, configura um servidor RPC e aguarda conexões de clientes.
- Cliente: Conecta-se ao servidor, registra-se, e inicia a interface gráfica, enviando comandos e buscando atualizações do estado do jogo periodicamente.

Problemas enfrentados e aprendizados

Durante o desenvolvimento, enfrentamos vários desafios que proporcionaram valiosos aprendizados:

1. Sincronização dos Estados: Garantir que todos os clientes tivessem uma visão sincronizada do estado do jogo foi um desafio significativo. Solução: Implementação de uma thread dedicada no cliente para solicitar atualizações periódicas do servidor.

2. Identificação de Jogadores: Inicialmente, todos os jogadores eram representados pelo mesmo símbolo, o que causava confusão. Solução: Atribuição de símbolos únicos ou cores diferentes para cada jogador, facilitando a identificação na interface gráfica.

3. Idempotência dos Comandos: Garantir que comandos duplicados não causassem inconsistências no estado do jogo. Solução: Implementação de identificadores únicos para cada comando e verificação no servidor para evitar a execução repetida de comandos processados.

Conclusões

A implementação de um jogo multiplayer utilizando Go e RPC foi um projeto desafiador e enriquecedor. Conseguimos transformar um jogo de jogador único em um ambiente multiplayer funcional, onde múltiplos jogadores podem interagir no mesmo mapa. Através da divisão clara das responsabilidades entre o servidor e os clientes, garantimos uma experiência de jogo sincronizada e consistente. As soluções desenvolvidas para os problemas enfrentados, especialmente em relação à sincronização de estados e identificação de jogadores, contribuíram para o sucesso do projeto. Este trabalho não só consolidou nossos conhecimentos em Go e RPC, mas também proporcionou uma compreensão mais profunda sobre a arquitetura de sistemas multiplayer.