# Documentação do Sistema de Jogo de Labirinto com Sockets

Arthur Linhares Madureira
Matrícula: 2021031599

1 de dezembro de 2024

## 1 Introdução

Este documento apresenta a documentação detalhada do sistema de jogo de labirinto desenvolvido em linguagem C. O sistema é composto por duas partes principais: o servidor, responsável por gerenciar o estado do jogo e processar as ações do cliente, e o cliente, que permite aos usuários interagirem com o servidor, enviar comandos e visualizar o estado do jogo. Este documento também aborda os principais erros encontrados durante o desenvolvimento e as soluções adotadas para superá-los.

## 2 Visão Geral do Sistema

O sistema de jogo de labirinto foi projetado para permitir que um cliente se conecte a um servidor central, interaja com o labirinto e jogue em tempo real. A comunicação entre o servidor e o cliente é realizada através de sockets, utilizando as versões IPv4 ou IPv6 conforme a configuração.

## 3 Servidor

### 3.1 Descrição Geral

O servidor é responsável por gerenciar o estado do jogo de labirinto, processar as ações enviadas pelos clientes e comunicar as atualizações necessárias. Ele utiliza sockets para escutar conexões de clientes e manter a comunicação ativa durante a sessão de jogo.

## 3.2 Componentes Principais

- **Socket de Escuta**: Configura e gerencia o socket que escuta por novas conexões de clientes.
- **Estado do Jogo**: Mantém a representação atual do labirinto, posição do jogador, e estado geral do jogo.
- **Manipulador de Ações**: Processa as ações recebidas dos clientes, atualizando o estado do jogo conforme necessário.
- **Algoritmo BFS**: Implementa o algoritmo de Busca em Largura para encontrar caminhos no labirinto, fornecendo dicas aos jogadores.

## 3.3 Principais Erros Encontrados e Soluções

Durante o desenvolvimento do servidor, vários desafios foram enfrentados. A seguir, descrevem-se os principais erros encontrados e as soluções adotadas:

### 3.3.1 Erro na Leitura do Labirinto

**Problema**: A função responsável por ler o labirinto a partir de um arquivo não estava tokenizando corretamente as linhas, resultando em uma matriz de labirinto malformada.

**Solução**: Revisou-se a lógica de tokenização utilizando `strtok`, garantindo que as linhas fossem divididas por espaços, tabulações e quebras de linha. Além disso, implementaram-se verificações para assegurar que os índices de linhas e colunas não excedessem os limites definidos por `MAX_SIZE`.

### 3.3.2 Configuração Incorreta do Socket

**Problema**: O servidor não conseguia vincular o socket à porta especificada, causando falhas na execução.

**Solução**: Identificou-se que a opção `SO_REUSEADDR` não estava sendo corretamente definida. Corrigiu-se a ordem das chamadas de `setsockopt` e `bind`, assegurando que o socket pudesse ser reutilizado imediatamente após o fechamento.

### 3.3.3 Inicialização Incorreta do Estado do Jogo

**Problema**: O servidor não estava inicializando corretamente o estado do jogo, resultando em posições erradas do jogador e da saída no labirinto. Isso levava a comportamentos inesperados, como o jogador não aparecendo na posição correta ou a saída não sendo reconhecida.

**Solução**: Implementou-se a função `initialize_game`, que percorre o labirinto para localizar as posições de entrada (valor 2) e saída (valor 3). Ao encontrar a entrada, define-se a posição inicial do jogador e marca-se essa posição no estado do jogo. Similarmente, a posição da saída é registrada. Além disso, a função `reveal_cells` foi criada para revelar as células ao redor do jogador, garantindo que o estado do jogo reflita corretamente a posição atual do jogador e as áreas visíveis do labirinto.

#### 3.3.4 Uso do Algoritmo BFS para Fornecer Dicas de Movimentos

**Problema**: O servidor precisava fornecer dicas de movimentos para os jogadores, indicando o caminho mais eficiente até a saída do labirinto.

**Solução**: Implementou-se o Algoritmo de Busca em Largura (BFS) na função `find_path`. O BFS foi utilizado para encontrar o caminho mais curto do jogador até a saída do labirinto. Ao localizar esse caminho, a sequência de movimentos foi armazenada e enviada ao cliente como uma dica.

# 4 Cliente

## 4.1 Descrição Geral

O cliente permite que os usuários interajam com o servidor, enviem comandos e visualizem o estado do jogo de labirinto. Ele se conecta ao servidor através de sockets, envia ações de jogo e recebe atualizações para manter a interface do usuário atualizada.

## 4.2 Componentes Principais

- **Socket de Conexão**: Estabelece e gerencia a conexão com o servidor.
- **Interface do Usuário**: Fornece uma interface de linha de comando para que o usuário interaja com o jogo.
- **Manipulador de Entrada**: Processa os comandos inseridos pelo usuário e os converte em ações para enviar ao servidor.
- **Gerenciador de Exibição**: Recebe e exibe as atualizações do servidor, como o estado do labirinto e dicas de movimento.

## 4.3 Principais Erros Encontrados e Soluções

O desenvolvimento do cliente também apresentou desafios significativos. A seguir, destacam-se os principais erros e as soluções implementadas:

#### 4.3.1 Conexão com o Servidor Falhando

**Problema**: O cliente frequentemente não conseguia estabelecer conexão com o servidor, especialmente ao utilizar diferentes versões de IP (IPv4 e IPv6).

**Solução**: Melhorou-se a função `setup_client_socket` para suportar tanto IPv4 quanto IPv6, utilizando a flag `AF_UNSPEC` no `getaddrinfo`.

#### 4.3.2 Recepção Incorreta de Dados

**Problema**: O cliente estava interpretando incorretamente os dados recebidos do servidor, resultando em representações errôneas do labirinto.

**Solução**: Ajustou-se a lógica de recebimento e processamento de dados, garantindo que a estrutura `action` fosse corretamente mapeada e que o número de linhas e colunas fosse adequadamente determinado antes de exibir o mapa. Para isso, elementos fora do tabuleiro foram mapeados com -1 no servidor e ignorados no cliente.

### 4.3.3 Validação de Movimentos Inadequada

**Problema**: Movimentos inválidos eram permitidos, levando a inconsistências no estado do jogo.

**Solução**: Implementou-se a função `is_move_possible` para verificar se um movimento solicitado pelo usuário estava entre os movimentos possíveis fornecidos pelo servidor. Isso evitou que movimentos inválidos fossem enviados e processados.

# 5 Conclusão

A construção do sistema de jogo de labirinto envolveu a implementação de um servidor robusto e um cliente intuitivo, ambos capazes de se comunicar eficientemente através de sockets. Com esse trabalho, foi possível exercitar, na prática, a programação com sockets.