Trabalho Prático 1 da disciplina de Redes

Lorenzo Carneiro Magalhães - 2021031505

Belo Horizonte, 22 de novembro de 2024

1. Introdução

Nesse trabalho prático, foi desenvolvido um sistema de comunicação local via internet para implementar um jogo no modelo cliente servidor. Nesse modelo, o servidor guarda um labirinto e realiza os movimentos e ações solicitados pelo cliente. Apesar dos desafios encontrados, foi possível implementar uma solução correspondente à detalhada no documento de explicação do trabalho.

2. Implementação

A implementação foi dividida em 4 estágios: construção dos sockets, implementação do labirinto e as funções associadas, integração do labirinto com a rede, correção de problemas e implementação da dica.

2.1. Construção dos sockets

A construção base dos sockets foi totalmente implementada a partir do vídeo do professor Ítalo, o qual foi colocado como uma das referências do trabalho. Dessa maneira, não houve grandes dificuldades na implementação básica dos sockets.

2.2. Implementação do labirinto

Em teoria, a implementação do labirinto é fácil e rápida. Porém encontrei algumas dificuldades e atrasei muito devido à falta de proficiência na linguagem C. Cometi diversos erros como pensar que *if(-1)* resultaria em *false* e não conseguir passar como parâmetro tipos como int[ ][ ] e char**. Assim, após pesquisar sobre muitos erros, fui relembrando o funcionamento correto da linguagem. Entretanto, demorou bastante para finalizar a implementação correta do labirinto e suas funções associadas.

Para utilização mais fácil do labirinto, criei uma estrutura chamada *Maze*. As demais funções referentes ao labirinto levaram em consideração essa estrutura.

```c
typedef struct {
   int visited[10][10];
   int board[10][10];
   int moves[100];
   int px;
   int py;
   int size;
} Maze;

// Estrutura Maze
```

```c
extern char NUM2CHAR[6];
extern char* NUM2MOVE[5];
int direction2num(char* str);
int is_move_legal(Maze maze, int new_x, int new_y, int consider_barrier);
void get_legal_moves(Maze* maze, int* legal_moves);
void read_maze(Maze *maze, char* filename);
int make_move(Maze *maze, int move_index);
void maze2string(Maze maze, char* buf);
int is_win(Maze maze);
void gen_board_unvisited(Maze maze, int unvisited_board[10][10]);
void reveal_all(Maze* maze);
void reveal_around_player(Maze* maze);

// Funções e variáveis para o funcionamento do labirinto
```

Com essa implementação, é necessário instanciar um objeto Maze no cliente, porém ele não realiza nenhum processamento do jogo. Esse objeto Maze no cliente é instanciado para possibilitar a utilização da função *maze2string*, que transforma o tabuleiro no formato de string.

2.3. Integração do labirinto com a rede

Realizar integrações sempre é trabalhoso e dessa vez não foi diferente: adequar todas as funções para estarem nos conformes da especificação é demorado (e tedioso). Mas após mudanças no funcionamento dos sockets para adaptar ao envio de um struct e criação de novos scripts implementando essa integração, foi possível realizar o trabalho conforme as especificações.

Esses novos scripts são *common_client.h/c*, *client_maze.h/c* e *server_maze.h/c*. Esses dois scripts são bem similares e implementam uma função principal, que chama *client/server.action_handler*. Essa função chamava outras funções baseado no código recebido para compreender a requisição recebida através do struct de recebimento e preparar o struct de envio.

É importante ressaltar que no cliente também foi necessária a implementação da função *client_pos_action_handler*, que lida com o processamento imediatamente após receber a resposta

Além disso, nessa fase percebi muitos erros relacionados à manipulação das matrizes e strings. Trabalhar com strings no C pode ser muito trabalhoso.

2.4. Correção de erros

Quando a integração base foi concluída, comecei a tratar os erros encontrados. Foram bastantes erros encontrados. Em sua maioria eram relacionados à interpretação das matrizes, sobrescrição de variáveis e vetores e manipulação de strings.

Pensando na sobrescrição de variáveis, decidi que era interessante criar no cliente e no servidor duas instâncias do struct *action*. Dessa maneira, conseguiria separar em um *struct* para receber dados e outro para enviar dados. Isso foi uma boa decisão, pois ajudou no fluxo.

Após essa correção de erros, trabalhei para deixar os *outputs* o mais fiel possível aos *outputs* encontrados na documentação. O processo foi demorado, mas creio que o output do meu programa está sempre igual ao do especificado.

2.2. Implementação da dica

Como todas as funções necessárias para a dica já estavam implementadas, resolvi realizar a função de cálculo de caminho mais próximo como dica. O algoritmo que usei para isso foi a BFS, que calcula o melhor caminho quando o peso das arestas são uniformes. Para facilitar a implementação, criei um novo struct chamado *Position*.

```
typedef struct {
   int x;
   int y;
   int parent_index;
   int direction;
} Position;
```

A única diferença para uma BFS normal foi o armazenamento da direção escolhida, que foi usada conjuntamente com a variável *parent_index* do struct para recuperar o caminho desejado.

3. Conclusão

O trabalho foi executado com sucesso, porém houve uma demora inesperada para a realização dele, devido aos problemas relatados ao longo da documentação.

Por se tratar de um documento curto, não é possível detalhar todas as decisões de implementação, mas creio que é o suficiente.

Por fim, achei muito interessante trabalhar com sockets e ressalto que a aula do Ítalo foi essencial para a realização do trabalho.