Universidade Federal
de São João del-Rei

Trabalho Prático 1

Dilvonei Lacerda
Joel Silva Campos das Chagas
Tomaz Miranda de Oliveira

Neste trabalho iremos implementar um jogo de truco baseado em turnos, para a disciplina Redes de Computadores do curso de Ciência da Computação da Universidade Federal de São João del Rei.

São João del-Rei

Outubro de 2022

Sumário

1 Introdução

Neste trabalho iremos implementar um jogo de truco baseado em turnos, no qual deve ser jogado por quatro participantes. Optamos por utilizar o protocolo TCP, pois é necessário ter garantia que um dado enviado pelo remetente realmente chegou ao destinatário. Já UDP é utilizado quando o volume de dados a ser transmitido é grande e não há necessidade de garantia de recepção de todos os dados.

1.1 Regras do Jogo

O Truco Mineiro deve-se jogar em duplas/pares ou trios (no caso de duplas ou trios, os jogadores devem alternar suas posições na mesa entre os parceiros e adversários).

Quem faz doze pontos [tentos] primeiro vence. O jogador/dupla/trio ganha a queda quando vence dois jogos.

Os participantes recebem três cartas a cada mão/rodada e disputam uma melhor de três. Uma carta é colocada na mesa por rodada, e a mais valiosa vence a vez.

Ao ganhar duas rodadas, o jogador/dupla/trio recebe os pontos da mão.

Em caso de empate na primeira rodada, a segunda decide o vencedor da mão. Em caso de empate na segunda ou terceira rodada, ganha quem venceu a primeira.

Em raros casos de empate em todas as rodadas, ninguém ganha os pontos da mão.

1.1.1 PONTUAÇÃO NO TRUCO MINEIRO

- - Cada mão começa valendo 2 pontos;
- - No decorrer das rodadas os jogadores, na sua vez de jogar, podem pedir truco, aumentando o valor da mão para 4 pontos;
- - Os adversários podem aceitar, rejeitar ou pedir 6. Quando rejeitam, a mão termina e quem pediu truco ganha 2 pontos. Caso o adversário peça 6, quem pediu truco tem as mesmas possibilidades, aceitar, rejeitar, ou aumentar para 10;
- - Os aumentos de apostas podem chegar até 12 e, em seguida, queda.

1.1.2 MÃO DE DEZ

É quando uma das duplas atinge 10 pontos. Nesta mão, os jogadores da dupla/trio com 10 pontos olham as cartas um do outro antes de jogar, decidindo se vão jogar ou não a mão.

Se jogarem, a mão vale quatro pontos. Se rejeitam, seus adversários ganham dois pontos.

1.1.3 MÃO DE FERRO

Em caso de empate em 10 a 10, as duas duplas disputam a mão às cegas. As cartas permanecem viradas de cabeça para baixo, ninguém pode ver suas próprias cartas.

Quem vencer a mão, ganha o jogo.

1.1.4 BARALHO TRUCO MINEIRO

O truco mineiro é jogado com 40 cartas, retirando os coringas, 8, 9 e 10. A sequência das cartas, independente do naipe, na ordem de mais valiosa para menos valiosa: 3 > 2 > A > k > J > Q > 7 > 6 > 5 > 4.

Manilhas são as cartas com maior valor no Truco. No Truco Mineiro, as manilhas são fixas.

O zap, 4 de paus, é a carta mais forte do Truco. Da mais valiosa para a menos valiosa, as manilhas no truco mineiro são: 4 de paus (Zap) > 7 de copas > A de espadas (Espadilha) > 7 de ouros.

2 Listagem das rotinas

A aplicação funciona como o modelo cliente-servidor. O servidor é responsável por organizar todos os recursos e todas as comunicações são trazidas através dele. Cliente envia mensagem (quando autorizado) para o servidor e o servidor é responsável por todo o processamento do jogo enviando um novo status para o jogador. Exemplo: O servidor envia uma mensagem a um jogador mencionando ser a sua vez de jogar. Ao mesmo tempo, envia mensagens para outros jogadores para que eles saibam quem está jogando. As tentativas de interação de outros jogadores são ignoradas. Quando um jogador confirmado se move, o sacador faz todas as inspeções (se a carta é a mais forte da mesa, se há empate, etc.). Assim que os quatro participantes se conectam, ocorre o embaralhamento das cartas e em seguida é realizada a distribuição das mesmas, assim como é apresentada na imagem abaixo.

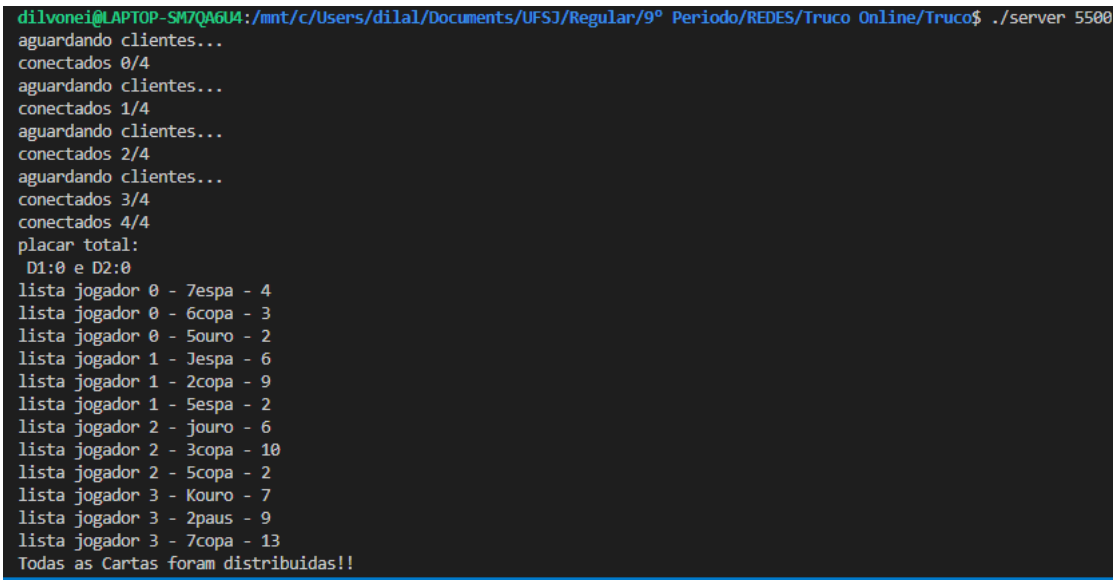

Figura 1: Log Servidor

Abaixo também podemos visualizar as cartas chegando a cada destinatário:

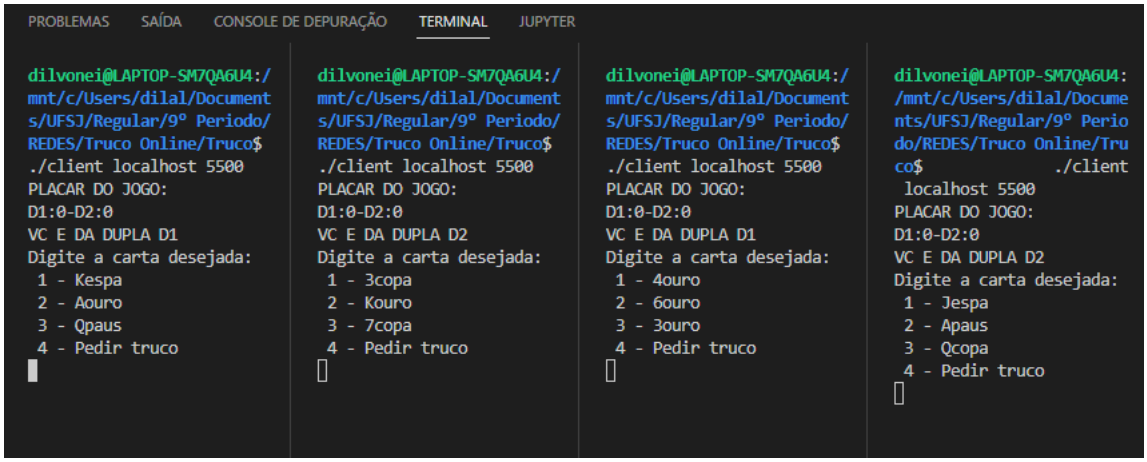

Figura 2: Recebimento de cartas

3 Descrição dos algoritmos e estruturas de dados utilizadas

3.1 Servidor

Este módulo é responsável por armazenar as regras do jogo e suas estruturas. Nele são administradas informações sobre descritores de arquivos sockets criados para cada um deles, a princípio a ideia central seria utilizar threads, porém não foi necessário devido ao modo de jogo, permitindo apenas uma pessoa jogar por vez.

3.1.1 socklen_t clilen

Está estrutura armazena o tamanho de cada endereço conectado ao servidor.

```
1    socklen\_t clilen[MAX_CLIENTS];
```

3.1.2 Struct sockaddr_in

Está estrutura armazena o endereço do servidor contendo, porta e ip, além disso, armazena um vetor contendo todas as conexões existentes.

```
1    struct sockaddr\_in serv\_addr, cli\_addr[MAX_CLIENTS];
```

3.2 Client

Este módulo é responsável por enviar mensagens para o servidor, sua função é apenas de receber e enviar mensagem, onde não é realizada nenhuma chamada de funções do jogo em si, pois o mesmo roda exclusivamente no servidor.

1. Cria um soquete TCP.
2. Conecte o soquete do cliente recém-criado ao servidor.
3. Recebe cartas do servidor.
4. Envia cartas para o servidor.

3.3 Truco

Este módulo é responsável por armazenar as regras do jogo e suas estruturas.

1. **atribui_nome():** Nomes de cartas que são números, é o próprio número, e as Q, J, K e A estão em maiúsculo.

2. **atribui_valor ():** Função responsável por atribuir um valor a cada carta, os valores atribuídos são referentes as regras de jogo.
3. **imprimir_carta():** Realiza a impressão de uma carta.
4. **imprimir_baralho():** Realiza a impressão do baralho.
5. **imprimir_mesa():** Esta função mostra as informações do jogo como rodadas, quantidade de pontos de cada duplas e quais cartas estão na mesa.
6. **imprimir_mao():** Realiza a impressão das cartas que cada jogador possuí em sua mão.
7. **cria_baralho():** Função para ser criado o baralho, chamando as funções "atribui_nome()"e "atribui_valor()", e assim gerando todo o baralho.
8. **embaralhar():** Função responsável por realizar o embaralhamento das cartas, completamente randômica.
9. **distribuir_cartas():** Esta função distribui cartas aos jogadores, uma a uma, até que o número de cartas de cada jogador seja 3 e o número total de cartas na mesa, seja 12.
10. **cria_mesa():** Cria a mesa salvando as cartas que foram jogadas pelos jogadores.
11. **maior_carta():** Função utilizada para identificar a maior carta, jogada na mesa naquela rodada, retornando o índice do jogador que a jogou.
12. **placar():** Utilizando a função acima, recebe o id do jogador que jogou a maior carta na mesa, verifica qual a sua dupla e atribui os pontos daquela rodada a eles.
13. **buscar_valor():** Função utilizada para buscar o valor de uma determinada carta.

3.3.1 Struct CARTAS

Está estrutura de dados contém o nipe de cada carta e seu valor dentro das regras do jogo.

```
1 typedef struct CARTAS{
2     char nome_nipe[TAM];
3     int valor;
4 }CARTAS;
```

3.3.2 Struct JOGADOR

Está estrutura de dados contém o id da dupla, um array de três posições do tipo CARTAS, que abriga as três cartas do jogador a cada rodada.

```
typedef struct jogador{
    int id_dupla;
    CARTAS carta[3];
}JOGADOR;
```

3.3.3 Struct BARALHO

Está estrutura de dados contém todas as cartas do baralho.

```
typedef struct baralho{
    CARTAS carta[MAX];
}BARALHO;
```

3.3.4 Struct MESA

Está estrutura é responsável pela administração do jogo, contendo as cartas jogadas a cada rodada, o placar de cada dupla e também obtém o ganhador de cada rodada e o número de rodadas define se o jogo chegou ao final.

```
typedef struct mesa{
    CARTAS carta[4];
    int placar_dupla[2];
    int id_dupla[4];
    int num_rodadas;
}MESA;
```

4 Protocolo desenvolvido

Desenvolvido visando facilitar a transmissão de dados de uma maneira segura e precisa, o protocolo TCP possui uma característica importante para e que foi um dos motivos para ser escolhido na implementação deste trabalho. A característica ponto a ponto, que significa que a conexão será sempre estabelecida entre dois pontos, no nosso caso, cliente e servidor.

Além disso, outro ponto positivo de tal protocolo, é a utilização de várias técnicas para proporcionar uma entrega confiável dos pacotes de dados. Fazendo com que, dependendo da aplicação, como no nosso caso, se faz mais vantajoso do que o protocolo UDP.

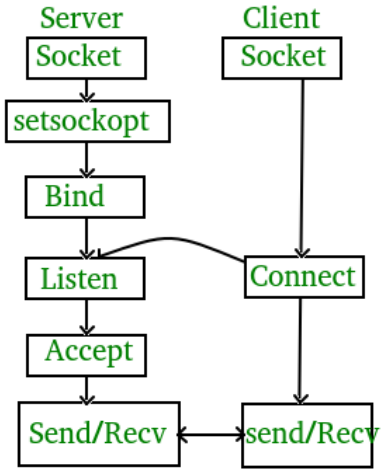

Figura 3: Conexão Client-Server

Todo o processo foi dividido nas seguintes etapas:

Servidor TCP -

1. Usando create(), criar socket TCP.
2. Usando bind(), vincule o soquete ao endereço do servidor.
3. Usando listen(), coloca o socket do servidor em modo passivo, onde ele espera que o cliente se aproxime do servidor para fazer uma conexão.
4. Usando accept(). Neste ponto, a conexão é estabelecida entre cliente e servidor, e eles estão prontos para transferir dados.
5. Volte para a Etapa 3.

Cliente TCP -

1. Crie um soquete TCP.
2. Conecte o soquete do cliente recém-criado ao servidor.

5 Conclusão

Neste trabalho, aprendemos de maneira prática, como funciona uma conexão cliente-servidor. Foi possível, também, aprender mais sobre o protocolo TCP, uma vez que o mesmo foi o escolhido para ser implementado e realizar a conexão citada acima.

Acreditamos que esse trabalho foi de suma importância para que a visualização mais prática de todo o conteúdo que havíamos aprendido em aula, fosse possível. Fazendo com que o mesmo, além de nos desafiar no quesito da programação, também nos agregou muito conhecimento sobre a parte implementada e como ela funciona.

6 Referências Bibliográficas

https://github.com/paulohtobias/ufsj-redes-tp1/

https://copag.com.br/blog/detalhes/truco-mineiro

https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-/
/clients-on-server-without-multi-threading/