Universidade Federal
de São João del-Rei

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
REDES DE COMPUTADORES 2024/1

IMPLEMENTAÇÃO DO JOGO YATZY MULTIUSUÁRIO EM C

Gustavo Henriques da Cunha

São João del-Rei
2024

Lista de Figuras

Sumário

1 INTRODUÇÃO

Este é um trabalho prático da disciplina de Redes de Computadores no curso de Ciência da Computação na UFSJ, tendo como docente o professor Rafael Sachetto Oliveira.

1.1 Objetivo

Neste trabalho, temos como objetivo implementar um jogo multiusuário em Unix, utilizando a comumicação entre cliente e servidor, onde no cliente teremos apenas o desenho da tela e a entrada de dados pelo usuário, e no servidor o tratamento da lógica do jogo e um intermediário entre os jogadores.

O jogo escolhido foi o Yatzy, um jogo de dados famoso em diversos países, onde recebe diferentes nomes, que consiste em uma disputa entre dois jogadores, que em turnos, buscam combinações de dados para preencher sua tabela de pontuação, buscando pontuar mais que o adversário.

1.2 Motivação

O trabalho é importante para praticar os conceitos introduzidos em sala de aula, buscando entender melhor o funcionamento das camadas de aplicação e transporte, além da programação em *socket*.

2 Yatzy

O jogo Yatzy, nome popular em países europeus, também chamado de Yatzee nos Estados Unidos e General aqui no Brasil, é um jogo de dados, multijogador, comumente dois, e de turnos, onde se busca uma maior pontuação que seu adversário para vencer.

Em cada turno, o jogador têm cinco dados, onde, tendo três chances, irá lança-los, podendo escolher qunatos dos dados manter. Ao final das três chances ele deverá escolher uma das 13 opções da tabela de pontuações e as faces dos dados darão sua pontuação, sendo que cada opção só pode ser usada uma vez. Além das opções da tabela também é possível conseguir um bônus.

A tabela de pontuação consiste das seguintes opção:

1. Uns - Consiste na soma de todos os dados mostrando um.

2. Dois - Consiste na soma de todos os dados mostrando dois.

3. Três - Consiste na soma de todos os dados mostrando três.

4. Quatros - Consiste na soma de todos os dados mostrando quatro.

5. Cincos - Consiste na soma de todos os dados mostrando cinco.

6. Seis - Consiste na soma de todos os dados mostrando seis.

7. Trinca - Soma de todos os cinco dados se pelo menos três forem iguais.

8. Trinca - Soma de todos os cinco dados se pelo menos quatro forem iguais.

9. Full House - 25 pontos por uma trinca e um par.

10. Sequencia Pequena - 30 pontos por uma sequência de quatro dados (1-2-3-4 ou 2-3-4-5 ou 3-4-5-6).

11. Sequencia Grande - 40 pontos por uma sequência de cinco dados (1-2-3-4-5 ou 2-3-4-5-6).

12. Yatzy - 50 pontos por cinco dados iguais.

13. Chance - Soma de todos os dados (qualquer combinação).

14. Bônus - Se a soma da pontuação total dos itens de um a seis for maior ou igual a 63, o jogador recebe 50 pontos extras.

Ao final de 13 turnos, ganha o jogador com maior pontuação. Em caso de pontuação igual, temos empate.

3 Programa

Para utilizar o programa os arquivos devem ser devidamente compilados.

Primeiro, deve ser executado o servidor, com a porta sendo passada na linha de comando. Depois, devem ser executados os clientes, em diferentes terminais, passando a mesma porta que foi passada ao servidor.

Para jogar o cliente deve esperar sua vez, e quando esta chegar, deverá seguir as instruções imprimidas no terminal e escolher números para identificar os dados que devem

ser mantidos e depois a posição da tabela que deve ser preenchida. A tabela será desenhada no terminal depois do término de cada escolha dos jogadores.

4 Implementação

O código foi todo implementado em C, modularizado para separar o cliente, o servidor e as funções do jogo. Foi utilizado as funções de *socket* padrões do C.

Foi seguido o modelo cliente-servidor, onde o servidor manda as informações ao cliente, o cliente imprime as informações na tela, recebe as entradas do usuário, que manda novamente ao servidor que cuidará da lógica do jogo, continuando o ciclo.

O jogo implementado é de dois jogadores por partida, mas se faz possível a realização de diversas partidas paralelas, através de um paralelismo no servidor. Assim, a cada dois jogadores conectados, se cria uma nova partida.

4.1 O Jogo

O módulo "yatzy"implementa algumas funções básicas para o funcionamento do jogo que são independentes do cliente e servidor. A maioria delas serve para desenhar no terminal, como as funções "*printInstructions*", "*printYatzyTable*"e "*printDice*", além de gerenciar a alocação e liberação da memória da tabela, com as funções "*initializeTable*"e "*freeTable*"e funções básicas sobre o funcionamento do jogo, que são "*rollDice*"e "*makeMove*". A figura 1 mostra como aparece no terminal as instruções feita pela função "*printInstructions*".

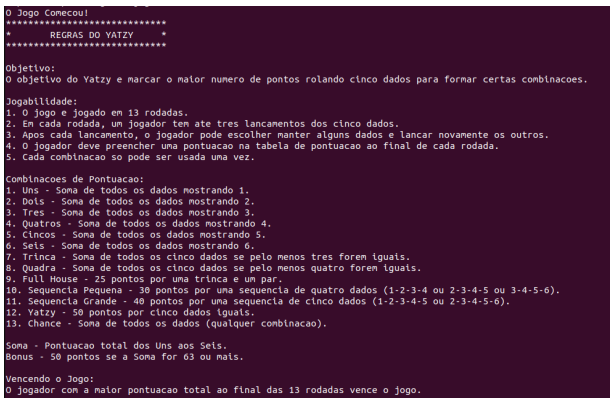

Figura 1: Instruções no terminal

4.2 Cliente

O cliente irá fazer a conexão com o servidor e fazer a comunicação através de mensagens, que serão detalhadas porsteriormente na seção do protocolo. Parte do código do cliente e do servidor foi feito com a ajuda do código do usuário nikhilroxtomar no GitHub.

As mensagens são feitas através de funções que enviam e recebem mensagens, tanto de texto como de números, com o servidor, como "*receive_int*"e "*receive_message*".

Além disso, o módulo "cliente"irá armazenar sua própria cópia da tabela de pontuação, que será atualizada pelo servidor sempre que houver modificações, pela função "*receive_update*", e imprimida no terminal. Também é papel do cliente receber as informações do usuário pelo teclado e encaminhá-las ao servidor.

4.3 Servidor

O modelo "servidor"implementa toda lógica do jogo, além da comunicação com os usuários. Este, junto com o protocolo de comunicação foi baseado no código do usuário beingaryan no GitHub.

Ele iniciará o *socket* e irá aguardar por conexões. Quando uma nova conexão chegar, verificará se ela ultrapassa o número máximo de conexões definido, e se existe outro cliente aguardando por uma partida. Caso sim, esses dois clientes serão pareados e uma nova partida começará. Caso não, ele terá que esperar por um novo cliente se conectar.

Com o uso de uma *thread* se faz possível ter conexões simultâneas, que geram uma condição de corrida, onde vêm a necessidade do uso de um *mutex*.

Quando dois clientes entram em uma partida, é feita as requisições e mandadas as informações necessárias, sendo a lógica do jogo sendo toda tratadas pelo servidor na função "*run_game*", que é disparada como uma *thread*. Nela é inicializado a tabela de pontuações, além de ser controlado os turnos dos jogadores, lançado os dados e calculado as pontuações, sendo toda informação enviada ao cliente para ser imprimida ao usuário.

4.4 Protocolo

O protocolo utilizado constiste de pequenas *strings* que o servidor sinaliza a ação que o cliente deve tomar. Depois de cada *string* o servidor irá esperar um número de inteiros, podendo enviar outros inteiros de volta.

A conexão irá se confirmar com o servidor mandando o id do cliente.
Depois, temos as *strings* usadas, que são:

- HLD - (*HOLD*): Sinaliza ao cliente para esperar o outro jogador se conectar.
- SRT - (*START*): Sinaliza ao cliente que o jogo irá começar. A comunicação efetiva entre cliente e servidor começa aqui.
- TRN - (*TURN*): Sinaliza ao cliente que é o seu turno. Depois disso, se repetirá o processo em que o cliente manda ao servidor os dados que ele escolheu manter, e o servidor manda os dados não mantidos depois de rolar eles mais uma vez, até que as três chances se acabem.
- CAT - (*CATEGORY*): Sinaliza ao cliente que escolha uma categoria da tabela.
- INV - (*INVALID*): Sinaliza ao cliente que sua escolha de categoria na tabela foi inválida.
- UPD - (*UPDATE*): Sinaliza ao cliente que se prepare para receber a tabela atualizada. Depois disso o servidor irá mandar o número de células da tabela a serem atualizadas, seguido pelas atualizações em si.
- WAT - (*WAIT*): Sinaliza ao cliente para esperar o turno do outro jogador.
- WIN - (*WIN*): Sinaliza ao cliente sua vitória.
- LSE - (*LOSE*): Sinaliza ao cliente sua derrota.
- DRW - (*DRAW*): Sinaliza ao cliente o empate.

5 Conclusão

Com o uso do conhecimento visto em aula, além de um aprofundamento na assunto, foi possível criar um jogo que aplica diversos conceitos de redes. No geral, o jogo funciona bem e cumpre com os requisitos estipulados anteriormente, mas algumas melhorias seriam necessárias para disponibilizá-lo em uso geral.

Um dos fatores importantes para seu uso seria um teste mais assíduo do programa, em diverentes máquinas e com diferentes números de usuários, além de uma mudança de

alguns detalhes e *bugs*, como quando o usuário digita algo quando é o turno do adversário, e acaba acrescentado lixo ao *buffer*.

No mais, para um projeto simples o programa funciona bem, apesar de ser uma implementação ligeiramente trabalhosa.

Referências

[beingaryan, 2020] beingaryan (2020). Tic-tac-toe-using-client-server-socket-programming-in-c. `https://github.com/beingaryan/TIC-TAC-TOE-USING-CLIENT-SERVER-SOCKET-PROGRAMMING-IN-C`.

[J. KUROSE, 2006] J. KUROSE, K. R. (2006). *Redes de Computadores e a Internet - Uma Nova Abordagem*. Addison-Wesley.

[nikhilroxtomar, 2017] nikhilroxtomar (2017). tcp-client-server-in-c. `https://github.com/nikhilroxtomar/tcp-client-server-in-C/blob/master/tcpServer.c`.