T1 - Laboratório de Redes e Computadores

Guilherme S. Rizzotto[1], João F. Leão[2]
Escola Politécnica— PUCRS

25 de maio de 2020

Resumo

O trabalho presente serve como método de avaliação da disciplina de Laboratório de Redes de Computadores ministrada pela professora Cristina Moreira Nunes como requisito para a obtenção do bacharelado em Engenharia de Software. O trabalho envolve a implementação de um jogo de perguntas e respostas respeitando a arquitetura cliente-servidor com troca de protocolos UDP.

1. Introdução

1.1. Visão Geral

Durante o quinto semestre do curso de Engenharia de Software, a disciplina de Laboratório de Redes de Computadores visa apresentar aos alunos diversos conceitos que abrangem a prática da disciplina de Redes de C000omputadores. O semestre apresentou diversos novos conceitos e revisitou alguns já estudados em diferentes cadeiras do curso. O trabalho presente serve como método de avaliação para identificar a aplicação desses conceitos em um software que deverá ser desenvolvido pelo alunos.

A seguir será descrito qual foi a problemática que motiva o desenvolvimento desse software, junto veremos como foi a implementação do mesmo, dificuldades, facilidades e escolhas arquiteturais que foram realizadas.

1.2. Enunciado

O enunciado do trabalho apresentava o problema de maneira simples. Considerando os conhecimentos adquiridos durante o semestre o grupo deveria criar um jogo. O desenvolvimento do jogo tem apenas duas regras de implementação, a primeira sendo a arquitetura cliente-servidor e a segunda utilização de *sockets* para protocolos UDP. O jogo deveria ser de perguntas e respostas sobre o conteúdo visto até agora em aula, as regras poderiam ser escolhidas pelo grupo.

[1]guilherme.rizzotto@edu.pucrs.br
[2]joao.leao@edu.pucrs.br

1.3. Concepção

O projeto foi primeiramente imaginado pelo grupo como um jogo individual em que o objetivo é acertar o maior número de perguntas (de preferência em sequência para aumentar mais a pontuação). O mesmo teria perguntas com dificuldade fácil e difícil e elas poderiam ser adicionadas por meio de um arquivo de texto no servidor. O jogo utilizaria o terminal para apresentar as perguntas e esperaria que o usuário enviasse a letra da resposta para receber um feedback.

Depois de criarmos um protótipo funcional dessa concepção inicial, ficou evidente que a usabilidade do jogo era de qualidade muito baixa, portanto o grupo decidiu implementar uma interface básica para melhor experiência do usuário. Como a versão final (com interface) reutilizou muitos métodos e trechos de código da versão inicial, as duas versões serão exploradas neste relatório.

2. Especificações

2.1. Tecnologias

O sistema foi desenvolvido na linguagem Java, uma linguagem de programação orientada a objetos. Na primeira versão do projeto (sem interface) o mesmo foi codificado na aplicação *Visual Studio Code*. Adiante no projeto, para criação da interface gráfica foi utilizado o *NetBeans*. O projeto se encontra em um repositório do *GitHub* que utilizamos para controle de versionamento.

No sistema são enviados pacotes através de *sockets* que utilizam O User Datagram Protocol (UDP) que não possui nenhum tipo de verificação de erros.

2.2. Arquitetura

O modelo de arquitetura utilizado foi cliente-servidor, a é uma estrutura que distribui as tarefas entre o fornecedor de recursos (servidor) e quem solicita o mesmo (cliente).

No nosso projeto, esse modelo de arquitetura se mostrou de grande importância, o acesso para o jogo seria feito pela área de cliente, enquanto o servidor seria responsável por verificar os dados e devolver um *feedback* em relação às questões respondidas. Na segunda versão do projeto a interface foi criada apenas para o cliente, uma vez que o servidor apenas responde as requisições recebidas.

2.3. Interface

Como indicado antes, na versão final do projeto decidimos implementar uma interface. A mesma só foi possível pela ferramenta *NetBeans*, que converte as telas criadas de forma interativa para código, facilitando e tornando mais rápido a criação das mesmas em Java.

Um dos principais motivos da criação da interface foi a experiência do usuário, que melhorou muito em relação a versão que utilizava apenas o terminal. Outra melhoria que foi observada graças a interface foi a manipulação de dados. Quando a aplicação contava com mensagens do usuário no terminal, qualquer mensagem errada, seja por sintaxe, formato ou acidente, poderia desencadear uma série de bugs no sistema. Depois de "limitarmos" as opções do usuário para que o mesmo fosse obrigado a selecionar botões predefinidos, os dados puderam ser manipulados e apresentados de maneira muito mais fácil, diminuindo o número de verificações de dados de entrada sem nenhuma perda de qualidade no software.

3. Implementação

3.1. Primeiros Passos

O primeiro passo do grupo para o desenvolvimento do software foi a criação dos pacotes Client e Server, dentro de ambos foram criadas classes principais com os nomes *Client.java* e *Server.java*. essas classes inicialmente contavam com um método *main()* que era responsável por fazer tudo, com o tempo novos métodos foram criados.

O próximo passo era garantir que sempre existisse uma conexão por meio de *socket* entre cliente e servidor. Essa conexão é iniciada nas primeiras linhas do método *main()* de ambas classe. Essa conexão é definida como variável global para que possa ser acessada por todos os métodos dentro da aplicação.

Depois de iniciar a conexão é necessário um método de conversa entre o cliente e servidor, para isso foram criadas os métodos *String receiveData()* e a classe *void sendData(String text)* em ambas as classe. Esses métodos serão explicados com mais detalhes na seção "3.5 - Fluxo de Dados Cliente-Servidor".

Para evitar que o relatório fique muito extenso não serão colocados trechos de códigos aqui, cada uma das classes e métodos foram documentados e estarão presentes no código fonte, onde está é possível visualizar melhor. Aqui serão citadas apenas a maneira que os mesmos contribuem para o funcionamento do jogo.

3.2. Servidor - Classes e Métodos

3.2.1. *Question.java*

Essa classe cria o objeto *Question*, o mesmo é utilizado para manter salvo todas as informações (id, pergunta, alternativas e resposta) de uma pergunta do jogo.

Além dos métodos de acesso *getters* e *setters* a classe possui um método *validateAnswer(int userAnswer)* que retorna um valor *booleano* indicando se resposta passada por parâmetro corresponde com a resposta da questão.

3.2.2. *Game.java*

Para tornar mais fácil o ato de iniciar um jogo essa classe foi criada para implementar o objeto *Game*. Esse objeto guarda uma lista com todas as perguntas de acordo com a dificuldade escolhida, uma lista com perguntas que já foram respondidas pelo jogador, pontuação do jogador e número de acertos seguidos do jogador.

Além dos métodos *getters* e *setters* de acesso, está implementado na classe o método *loadQuestions(String difficulty)* esse método abre o arquivo referente a dificuldade escolhida (easy.txt ou hard.txt) e adiciona na lista de perguntas os objetos *Question*, referentes aos dados carregados.

Outro método implementado é o método de sorteio de uma pergunta *getRandomQuestion()*. Para que o mesmo funcione, é sorteado um número e a pergunta com o id igual a esse número é retornada. Quando as questões acabam, um objeto *null* é retornado.

O último método relevante para essa classe é o *answerQuestion(Question a, in answer)*. Nesse método, é primeiramente verificado se a questão foi respondida corretamente, nesse caso, a pontuação do jogador aumenta em 10 + "modificador de sequência"pontos, e o "modificador de sequência" aumentado em 1 . Caso contrário, a pontuação permanece a mesma e o modificador volta para 0.

3.2.3. *Server.java*

Como explicado no fluxo cliente-servidor, as ações do cliente são enviadas por mensagens UDP contendo uma número e outros dados, esse número corresponde ao método que deve ser chamado. Depois da verificação do número pela classe *main()* o trecho de código referente é executado. Os possíveis métodos são os seguintes:

Para finalizar o server e voltar ao menu principal do jogo, o seguinte formato de mensagem deve ser enviado do cliente para o servidor

```
00
```

Essa linha indica que apenas o server deve parar de ser executado. Os adquiridos pontos até então, aparecerão na tela uma vez que o jogo finalizar o servidor, seja de maneira interrompida como a vista acima, ou quando todas as perguntas da dificuldade escolhida forem respondidas).

Para executar a criação de um novo objeto Game, o servidor deve receber uma mensagem com o seguinte texto.

```
01
true
```

A primeira linha apresenta o código utilizado para iniciar um jogo, a segunda linha indica qual dificuldade foi escolhida. Como o jogo possui as

dificuldades fácil e difícil, foi utilizado um booleano para que seja definida. Depois de criar o jogo, é executado o método que popula a lista de questões que podem ser respondidas.

Para que o cliente receba uma pergunta o mesmo deve enviar o seguinte formato de texto para o servidor.

```
1 02
```

A primeira linha apresenta o código que representa o sorteio de uma pergunta. A primeira coisa que o método faz é solicitar uma pergunta aleatória para o objeto *Game*, caso a pergunta seja *null* as perguntas acabaram e uma mensagem de finalização do jogo é enviada para o cliente. Caso contrário, a pergunta sorteada é enviada.

A penúltima tem o objetivo de informar qual alternativa o usuário escolheu.

```
1 03
2 1
```

A primeira linha indica que a alternativa escolhida para a última pergunta enviada está por vir. A segunda linha mostra qual alternativa foi escolhida pelo usuário (sendo 0 = a, 1 = b, 2 = c). Depois de realizar a leitura da resposta dada, é executado o método *answerQuestion(Question question, int answer)* do objeto Game. O retorno (boolean) junto da pontuação atualizada é enviado para o cliente.

Os métodos que tornam possíveis a troca de mensagens (receivers e senders) serão explicados na seção “3.5 - Fluxo de Dados Cliente-Servidor”.

3.3. Cliente - Métodos e Interface Gráfica

Pelo fato do cliente apenas solicitar as informações para o servidor, não foi necessário implementar nenhuma outra classe. Essa classe principal havia sido desenvolvida inicialmente pelo *Visual Studio Code* sem uma interface gráfica. Esse métodos foram alterados com o tempo para garantirmos que os dados seriam enviados mesmo que houvesse a perda de pacotes. Como já mencionado anteriormente, esses métodos de troca de mensagens serão explicados na seção “3.5 - Fluxo de Dados Cliente-Servidor”.

Durante o desenvolvimento da primeira versão da aplicação todas as manipulações de dados dos usuários eram realizadas no método *main()*. Assim que foi decidido criar uma interface, todas as funções tiveram que ser espalhadas em novos métodos para que ficasse funcional.

Como o desenvolvimento da interface foi todo realizado pela ferramenta *NetBeans*, grande parte do código que se refere a mesma foi gerado automaticamente. O motivo dos nomes dos métodos referentes aos botões serem diferentes do nome dos métodos que implementamos é o mesmo. Quando itens

interativos são adicionados na tela, métodos são criados na classe *Client.java* esses métodos têm nomes predefinidos. A partir desse momento, a interação com esses itens se tornam uma condição, onde o ato do clique desencadeia todos os trechos de códigos para o funcionamento do mesmo.

Alguns dos métodos que foram criados e exemplificam muito bem essa abordagem são os que precisam enviar algum dado para o servidor. Em geral temos três métodos assim:

O primeiro é responsável por iniciar um jogo novo. O envio é acionado quando o jogador pressiona o botão "Jogar" depois de ter selecionado uma dificuldade. Como apresentado acima, os dados são recebidos pelo servidor que toma conta de toda a criação do jogo;

O segundo é responsável por sortear uma pergunta. Ele é acionado quando o usuário pressiona o botão "Sortear Pergunta", a mesma recebe um retorno que é a pergunta junto das alternativas.;

O terceiro seria responder uma pergunta. O mesmo é acionado quando o botão "Enviar" é pressionado, logo depois de uma alternativa ser selecionada, o resultado vem junto com a pontuação do jogador nesta rodada, para que o mesmo possa estar sempre atualizado.

Por mais que existam muitos outros métodos, nem todos precisam ser explorados aqui, por isso, como indicado anteriormente, os mesmos estarão devidamente documentados no código fonte junto deste relatório.

3.4. Fluxo de Dados Cliente-Servidor

Para resolvermos os conflitos de recebimento e envio de dados, os trechos de código foram sincronizados de maneira que não haja conflitos. Os eventos principais (receber pergunta, enviar resposta, consultar pontuação) são controlados pelas ações do usuário e geram o envio de pacotes através de *Sockets* que utilizam o protocolo UDP.

Como o UDP não possui nenhuma verificação de erros e nenhuma confirmação de envio de dados, foi necessário implementar métodos de conversa. Esses métodos em sincronia enviam os pacotes aos seus destinos diversas vezes e são responsáveis por recolher eles para concluir uma troca de dados.

Baseando-se nas mensagens que são recebidos na classe Servidor (citada anteriormente) decidimos melhor ilustrar a lógica que o grupo seguiu. O diagrama a seguir apresentará as mensagens trocadas do servidor e do cliente em ciclo de um jogo com apenas uma pergunta.

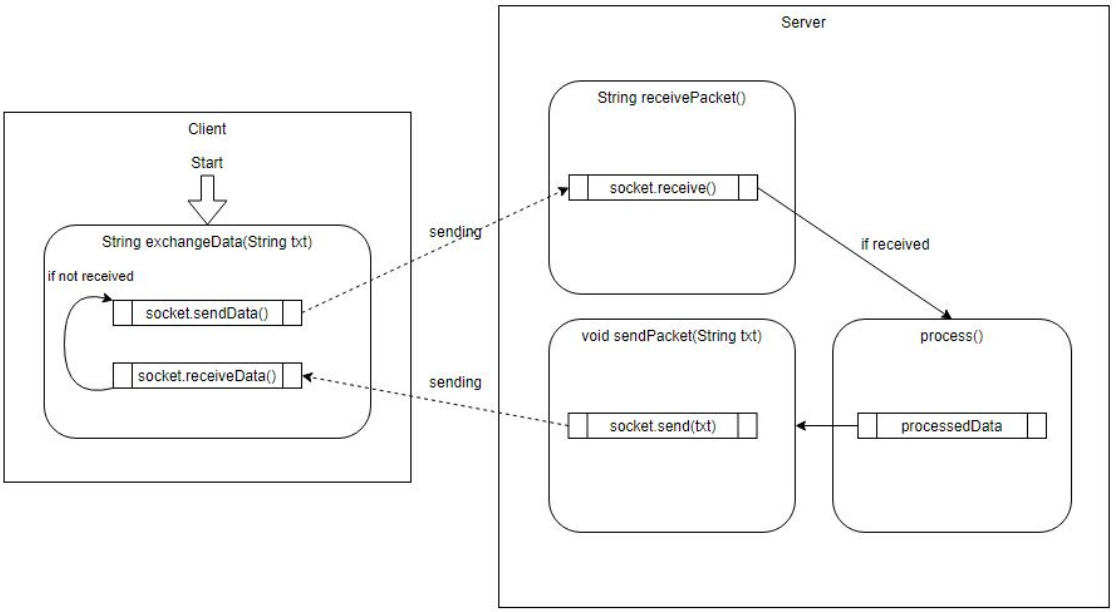

Figura 1: Diagrama de fluxo de dados

Como pode ser observado na imagem acima, enquanto o servidor e o cliente estão sendo executados, ambos ficam no aguardo até que a primeira mensagem (ação) seja reconhecida pelo servidor. Assim que o cliente é iniciado, ele começa a tentar enviar uma ação para o servidor, como o *socket* cliente foi definido junto de um timeOut de 10 milissegundos há uma tentativa de enviar e receber dados a cada 10 milissegundos.

Mesmo testando com níveis de perda de pacotes acima de 90%, os dados ocasionalmente chegam no servidor, quando isso acontece, eles são recebidos, tratados e processados devidamente. Depois de de cumprirem com a sua função, novos dados são enviados para o cliente, servindo de condição de parada para o envio incessante do mesmo antigo pacote.

Durante o desenvolvimento desse sistema, observamos que há uma maneira de "garantir" que os pacotes cheguem em algum momento, porém devido a capacidade de processamento de alguns computadores, consideramos que seria melhor criar uma condição de parada, tornando a aplicação mais responsiva diante de erros no fluxo dos pacotes UDP. Para resolvermos isso criamos um contador, o envio de pacotes pelo cliente não permanece ativo permanentemente. Caso depois de 50 tentativas de envio, nenhuma venha a resultar em retorno, a interface do cliente avisa o usuário que há um possível problema na rede, o indicando verificar o servidor e tentar novamente. Sempre que um protocolo é reenviado, o mesmo apresenta uma mensagem no *log* do cliente, alertando o programador de que os pacotes estão com alguma dificuldade para chegar do outro lado e voltar.

4. Resultados

Abaixo encontram-se as telas do projeto através da interface criada pelo *NetBeans*.

Figura 2: Tela inicial do projeto

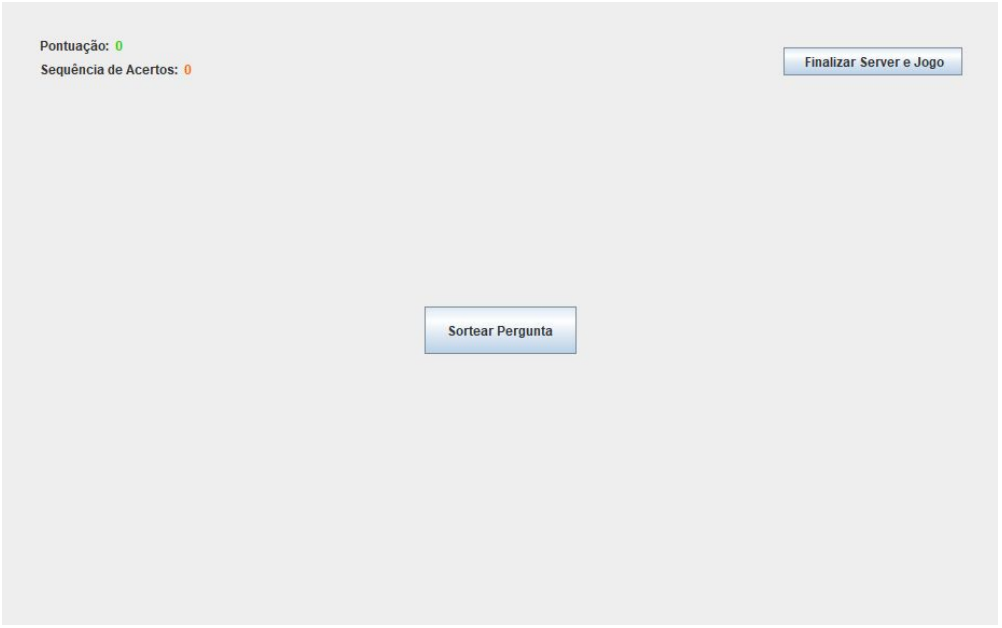

Figura 3: Tela para sortear uma pergunta

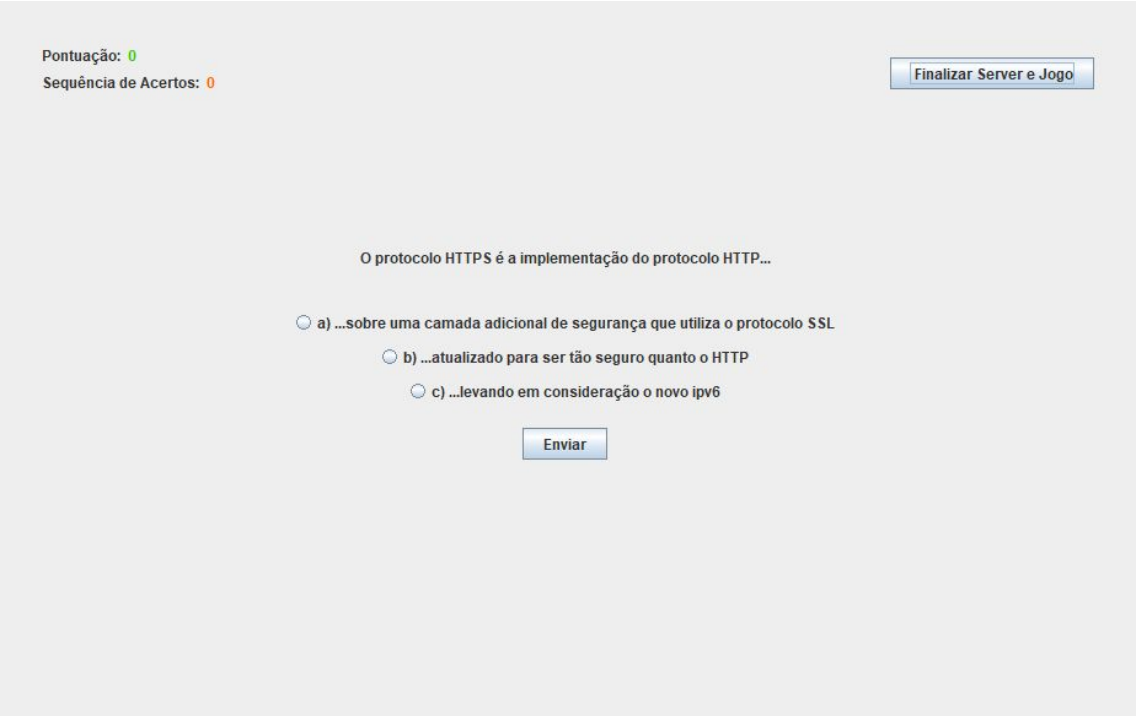

Figura 4: Tela que mostra a pergunta possibilitando responder

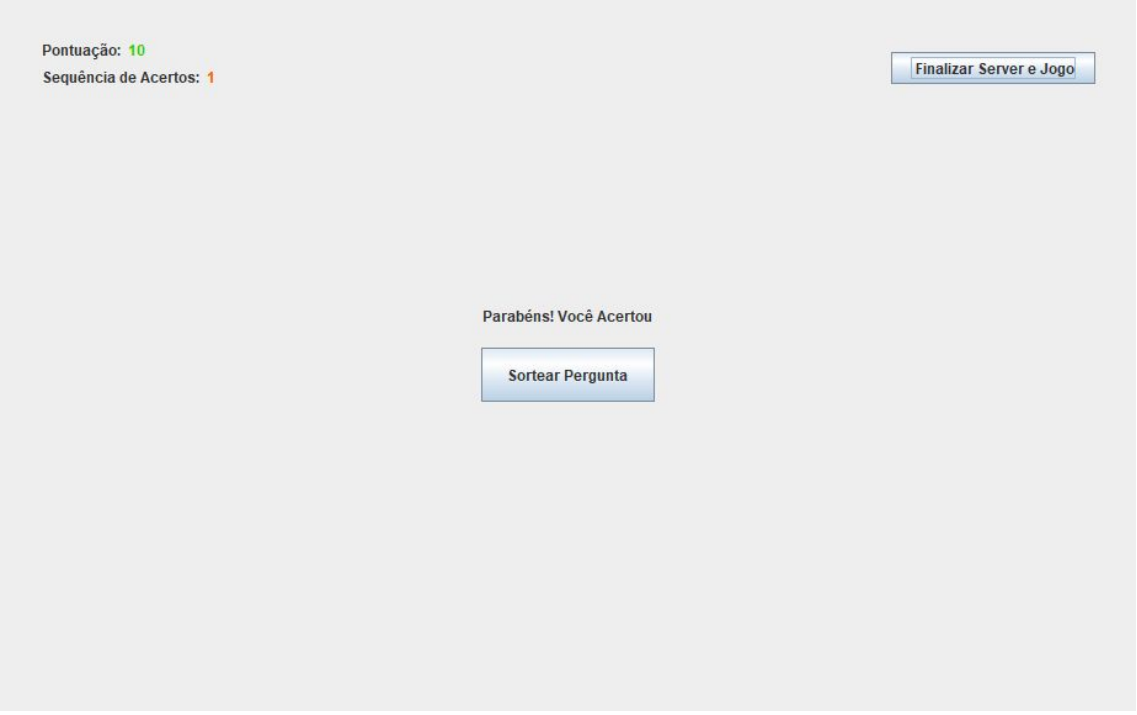

Figura 5: Tela de acerto

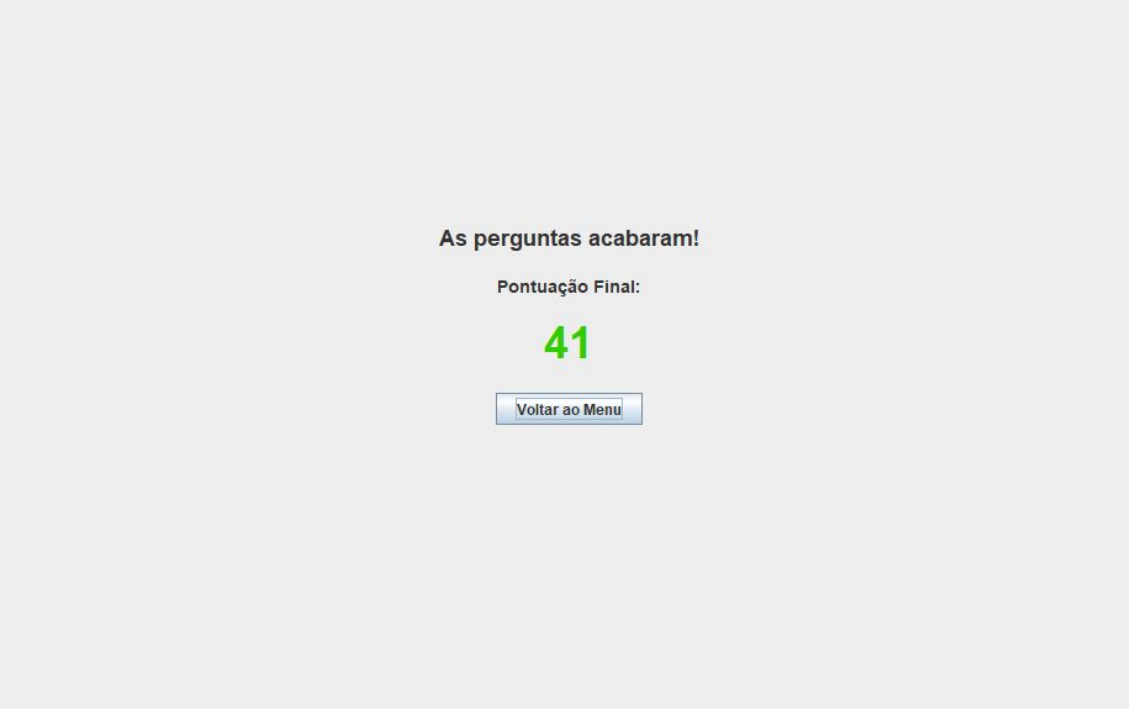

Figura 6: Tela final do projeto

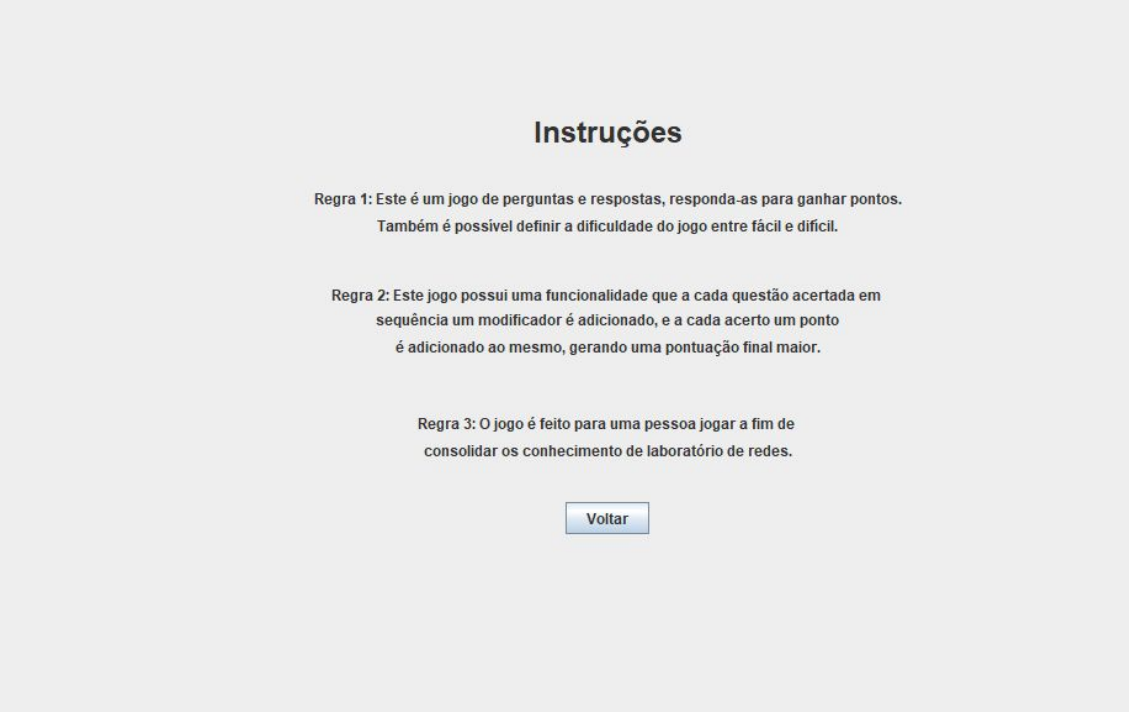

Figura 7: Tela de instruções do Jogo

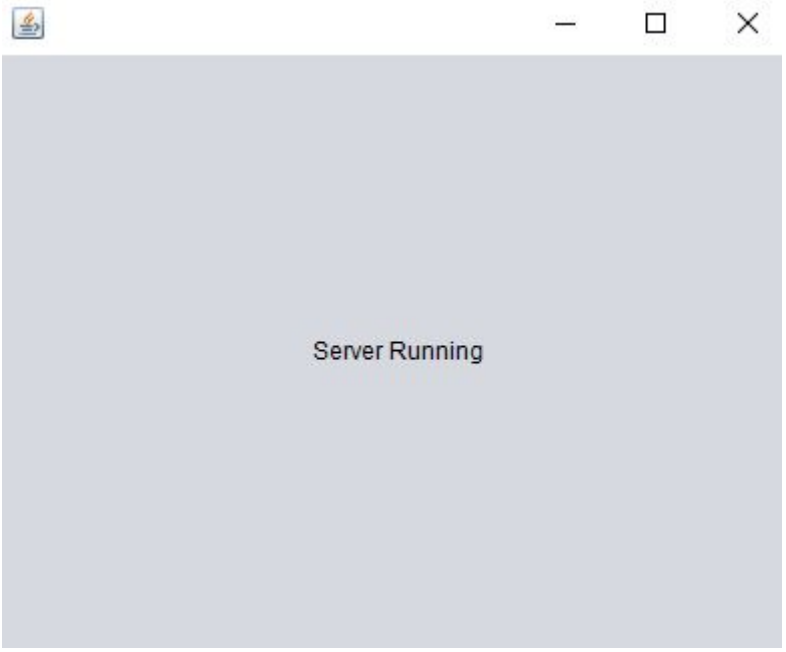

Figura 8: Tela de execução do server

5. Regras do Jogo

O jogo possui algumas regras estabelecidas pelos integrantes do grupo, elas são:

Regra 1 - Este é um jogo de perguntas e respostas, responda-as para ganhar pontos. Também é possível definir a dificuldade do jogo entre fácil e difícil.

Regra 2 - O jogo é feito para uma pessoa jogar a fim de consolidar os conhecimento de laboratório de redes.

Regra 3 - Este jogo possui uma funcionalidade que a cada questão acertada em sequência um modificador é adicionado, e a cada acerto um ponto é adicionado ao mesmo, gerando uma pontuação final maior.

6. Como Executar o Projeto

Para executar o projeto, dentro da pasta *LABREDES_CLIENT* e da pasta *LABREDES_SERVER,* existe uma pasta chamada dist, que contém um executável *.jar* a fim de facilitar a execução. Possuindo uma versão java compatível, execute os dois *.jar*. Caso algo não funcione, existem outros dois métodos para rodar a aplicação: com o NetBeans é possível compilar e executar o cliente e o servidor, ou no VSCode é possível utilizar o método run, que executa os dois.

7. Conclusão

Este trabalho foi de suma importância para entender os conceitos de *sockets* UDP e a maneira que funcionam na prática. Através da ferramenta clumsy, foi possível analisar a perda de pacotes e criar uma maneira de contornar isso, gerando o envio dos

mesmos novamente. Também, este trabalho possibilitou o aprendizado de como funciona a criação de interfaces em Java, visto que a dupla optou por criar algo visual para gerar uma experiência de usuário melhor.

Referências

Wireshark - Stable Release (3.2.2) February 26, 2020 - disponível em <https://www.wireshark.org/> último acesso 23/05/2020

Visual Studio Code - User Setup (1.45.1) May 14, 2020 - disponível em <https://code.visualstudio.com/> último acesso 23/05/2020

NetBeans IDE - Development Version (8.2 Patch 2) - disponível em <https://netbeans.org//> último acesso 23/05/2020

clumsy (0.2) - disponível em <https://jagt.github.io/clumsy/> último acesso 23/05/2020