Universidade de São Paulo - USP
Escola de Artes, Ciências e Humanidades - EACH
ACH2026 - Redes de Computadores - Turma 04
Professor João Bernardes

EP 1
~~O MELHOR JOGO DA VELHA QUE VOCÊ JÁ VIU~~
Utilizando arquitetura Cliente-Servidor

Débora Atanes Buss	nUSP: 9276860
Lucas Pipa Cervera	nUSP: 8094403
Vinícius Chaim	nUSP: 8585731

São Paulo
2018

1. PROPOSTA

A proposta é a criação de uma plataforma (na estrutura CLiente-Servidor) onde será possível jogar o popular *Jogo da Velha* (para mais informações e regras acesse aqui). De forma geral, conseguimos resumir o uso da aplicação da seguinte maneira:

- Ao entrar na plataforma (acessada via prompt de comando) você se identifica por um nome de usuário (único) e pode escolher dentre as três opções:

    1) *Jogo existente* - O Servidor passa uma lista de jogos existentes para que o usuário selecione um. Caso ele seja o primeiro jogador a entrar na partida, ele se torna o segundo jogador e jogo começa. Caso contrário ele pode assistir a partida via streaming.

    2) *Começar um novo jogo contra outro jogador* - O Servidor cria uma nova partida e aguarda até outro jogador entrar nela para iniciá-la.

    3) *Começar um novo jogo contra um bot* - O jogador pode treinar contra um bot controlado pelo Servidor.

- Também existe a opção de realizar *logout* para encerrar sua sessão.

- Nomes de usuários repetidos ou em branco não são permitidos para cada jogador online e o início de sua sessão é barrado até que seja fornecido um nome de usuário válido. Quando o usuário se desconecta (realiza logout ou tem sua conexão com o servidor quebrada), o nome que usava é liberado para ser utilizado por outras pessoas.

- O Servidor é o responsável por gerenciar os usuários e partidas ativas, autenticando novos usuários, liberando nomes de usuário quando os eles realizam logout, validando movimentos de usuários em partidas e repercutindo os estados das partidas ao espectadores da mesma.

OBS: Para essa entrega apenas estão disponíveis as funcionalidades de *login* (com validação de nome de usuário) e *logout*. Após o login há um menu interativo funcional (cada item selecionado é enviado ao servidor, o que pode ser verificado pelos logs recebidos no servidor), porém eles não realizam as ações propostas, ao invés disso retornam ao menu com a resposta do servidor (apenas a opção de *logout* do menu funciona).

2. ESPECIFICAÇÃO DA COMUNICAÇÃO

Utilizamos uma arquitetura cliente-servidor básica, utilizando o Socket do Java, que utiliza o protocolo TCP. Na classe Server, é onde iniciamos o servidor básico ouvindo a porta definida como padrão em uma constante, e que começa a ouvir as requisições que chegarem para si. Ao receber a requisição, caso seja uma nova conexão, o Server cria um ClientHandler, classe responsável por lidar com o recebimento e envio de mensagens para aquele cliente específico com o servidor como intermédio, e o adiciona a um vetor com todos os clientes quando o usuário realiza o login.

Utilizamos as classes do Java *ObjectInputStream* e *ObjectOutputStream* para transmitir um objeto *Request* customizado, com as informações que definimos como necessárias para uma boa comunicação entre o cliente e o servidor. Dentro desse objeto *Request*, colocamos um atributo status, do tipo *int*, onde controlamos a ação que está sendo tomada pelo cliente ou servidor, e para transmitir as informações necessárias. O objeto *Request* precisou ser um objeto serializado para ser transmitido via os *ObjectStream*.

Nesta etapa do exercício, ainda não implementamos criptografia, senhas ou timeout. A segurança é feita através da validação de usuários com identificação única (nome de usuário único). Após o cliente fazer a primeira conexão com o servidor, ocorre a tentativa de login, que consiste do Cliente mandar um nome de usuário dentro do objeto *Request* para o servidor, que o avaliará e, caso já exista um usuário conectado com aquele nome, retornará o status de erro, pedindo para que o cliente forneça um novo nome de usuário. Apenas após o usuário ser visto como válido pelo servidor que outras requisições além da tentativa de login são aceitas. Isso evitará falhas de consistência com os dados dos usuários online quando as demais funcionalidades forem implementadas e os usuários começarem a disputar e assistir partidas. Além disso, encerramos a sessão do cliente caso ele peça explicitamente para ser realizado logout ou caso ocorra alguma exceção com o *Client* e sua execução pare de rodar (o servidor captura isso e faz o *logout* daquele cliente).

3. GUIAS

Você pode adquirir o código fonte na pasta compactada que submetemos junto com este relatório ou clonar o projeto diretamente nesse repositório do github (Esse relatório é referente à **release v1.0**).

3.1. GUIA DE INSTALAÇÃO

Para rodar o código fonte, entre na pasta "*Jogo da Velha*" e siga os seguintes passos a seguir para compilar e executar cliente e servidor. Tenha em mente que para a aplicação funcionar você deve ter **um** servidor e **um ou mais** clientes (situados na mesma máquina ou em máquinas diferentes).

3.1.1. SERVIDOR (Você precisa de apenas um destes).

```
javac ./ServerPackage/Server.java
java ServerPackage/Server
```

3.1.2. CLIENTE (Rode quantos quiser e lembre-se de alterar <SERVER_IP> pelo IP do seu servidor. Não é necessário passar a porta como argumento, ela já se encontra configurada tanto no cliente quanto no servidor).

```
javac ./ClientPackage/Client.java
java ClientPackage/Client <SERVER_IP>
```

3.2. GUIA DE LEITURA DO CÓDIGO FONTE

O código fonte está dividido em 3 pacotes (Models, ServerPackage e ClientPackage) com os seguintes scripts/funções:

1) *Models* - Contém modelos que definem padrões a serem seguidos tanto no servidor quanto no cliente para uma boa comunicação. Possui:
    a) *Constants.java* - Armazena as constantes da aplicação, como o socket do servidor, valor dos status de mensagem, etc.
    b) *Requests.java* - É o objeto usado para comunicação entre cliente e servidor. O conteúdo de cada mensagem é mapeado em seus atributos

para que cliente e servidor tenham um modelo do que deve ser enviado/será recebido.

c) *GameModel.java* - Modelo esperado pelo cliente que o servidor envie informando a situação da partida (quando um jogo está acontecendo). Ainda não foi usado nessa versão. Posteriormente será adicionado a um atributo na classe *Requests*.

2) *ServerPackage* - Contém os códigos exclusivamente utilizados pelo Servidor para estabelecer a conexão com o cliente e lidar com o envio, recebimento e tratamento de mensagens. Contém:

a) *Server.java* - Responsável por sustentar/gerenciar o serviço. Cria Threads (do tipo *ServerHandler*) para lidar com cada cliente que deseja se conectar ao servidor e mantém controle de todos os clientes e partidas ativos. Através de seus métodos que é possível adicionar, gerenciar ou remover clientes (login, logout) e partidas (validações e propagação da informação aos outros usuários envolvidos com a partida).

b) *ServerHandler.java* - É o responsável por lidar com a comunicação com um único cliente, recebendo suas mensagens, encaminhando tratamentos e enviando as respostas.

3) *ClientPackage* - Contém os códigos exclusivamente utilizados pelo Cliente para estabelecer a conexão com o servidor. lidar com o envio, recebimento e tratamento de mensagens e como exibir as informações para o usuário final. Possui:

a) *Client.java* - Responsável por iniciar a comunicação com o servidor e coordenar a sessão do cliente, enviando as requisições do usuário e recebendo e interpretando as mensagens enviadas pelo servidor para decidir as ações a serem tomadas. Usa os métodos do *Navigator* para atualizar a navegação do usuário conforme interpreta as mensagens recebidas pelo servidor.

b) (UserInterface) Navigator.java - Responsável por lidar com a seleção de quais telas serão exibidas para o usuário (juntando fragmentos de *Interfaces* para organizar as informações para o usuário e traduzindo as mensagens recebidas em formas mais amigáveis de exibição) e com o

tratamento de inputs, que envia de forma “mastigada” para o *Client* enviar diretamente ao servidor;

c) *(UserInterface) Interfaces.java* - Possui os “fragmentos” de exibição de tela, usados para comunicação com o usuário. Lida APENAS com a saída de dados visual.

4. TESTES E RESULTADOS

Por enquanto foram realizados apenas testes reais relacionados ao gerenciamento de usuários (login/logout), pois apenas essa funcionalidade está completa até o momento. Porém, demais testes (que não constam abaixo) foram realizados com o intuito de verificar se o servidor estava recebendo e conseguindo trabalhar com todos os status de requisição realizados pelo cliente (até o momento) e o resultado foi positivo, as requisições estão sendo realizadas, apenas falta implementar os métodos que as trabalham devidamente (É possível observar o recebimento das requisições executando o Servidor e vendo seus *logs*, que trazem qual STATUS foi recebido por qual usuário/player. A resposta dessas requisições está definida para ser sempre a mesma, que no cliente provocará a reação de voltar ao menu (por isso que a cada requisição que não seja de SAIR o menu é reiniciado)).

4.1. GERENCIAMENTO DE USUÁRIOS:

- Mais de um cliente tenta fazer login com nome de usuário único - SUCESSO - Ambos os clientes foram logados no servidor e podiam interagir (enviar/receber mensagens).
- Mais de um cliente tentando fazer login, sendo que pelo menos um deles tentou usar um nome de usuário já usado por outro cliente - SUCESSO - O cliente teve seu login negado e foi requisitado que usasse um novo nome. Os demais clientes já logados não sofreram alterações.
- Cliente tentando fazer login com nome de usuário vazio - SUCESSO - O cliente teve seu acesso negado e foi requisitado que usasse um novo nome.
- Cliente tentando realizar logout - SUCESSO - O logout foi realizado, o cliente foi encerrado e o servidor continuou funcionando com os demais clientes online.

- Cliente tentando fazer login com o nome de usuário de um cliente que havia acabado de fazer logout - SUCESSO - Usuário conseguiu fazer login com sucesso, uma vez que não havia usuário online com o mesmo nome de usuário que desejava.

5. PROBLEMAS

Os principais problemas foram com barreiras técnicas do grupo por nunca termos implementado uma estrutura cliente/servidor usando sockets em Java anteriormente, principalmente no que se trata da serialização dos objetos a serem passados entre cliente/servidor e qual seria a melhor estrutura desse objeto para que fosse completo o suficiente para poder acolher todas as informações necessárias de serem recebidas por cliente e servidor ao mesmo tempo que não se tornasse uma classe gigante e confusa de ser interpretada por ambos.

As barreiras técnicas foram solucionadas com o tempo graças a materiais de programadores mais experientes no ramo que disponibilizaram seu conhecimento na internet (fontes que destacamos abaixo).

As dúvidas em relação a melhor estrutura de pacotes, classes e objetos foram sendo solucionadas da mesma forma que dissemos no parágrafo anterior, mas também (e principalmente) de forma empírica ao precisar enviar e receber as requisições por Cliente e Servidor e ir ajustando os modelos conforme a necessidade.

6. FONTES

1. https://www.geeksforgeeks.org/introducing-threads-socket-programming-java/

2. https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

3. https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html

4. https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html

5.https://stackoverflow.com/questions/14551211/program-pauses-on-initializing-object-input-stream-in-java