Jogo Batalha Naval em Rede com Java Sockets UDP

Israel Deorce Vieira Júnior[1]

[1] Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre, RS - Brazil

{israel.deorce@acad.pucrs.br}

Resumo. *Este relatório descreve uma alternativa de implementação para o exercício proposto no primeiro trabalho da disciplina de Laboratório de Redes. O exercício envolve o desenvolvimento de um jogo que funcione em rede e utilize* $sockets$*, devendo ser criado no mínimo um* $socket$ *servidor e um* $socket$ *cliente que devem enviar mensagens para controlar o jogo. Desenvolvemos a solução proposta implementando o tradicional jogo* $BatalhaNaval$ *utilizando a linguagem de programação Java orientada a objetos e* $sockets$ $Datagrama$ *(UDP), com auxílio de estruturas de dados do tipo matriz para representar o tabuleiro do jogo. Apresentaremos a solução de forma textual explicativa, acompanhada de pseudocódigo e imagens ilustrativas.*

1. Introdução

A programação de jogos seguindo o modelo cliente/servidor permite a criação de aplicações com interação de diversos usuários em uma rede, tornando possível o estilo de jogo com multijogadores. Para isso, é preciso entender bem os conceitos de $sockets$, protocolos de transporte (e.g., TCP e UDP) e como implementá-los.

A atividade proposta consiste no desenvolvimento de um jogo que funcione em rede e utiliza $sockets$, devendo ser criado no mínimo um socket servidor e um socket cliente que envia mensagens para controlar o jogo. A solução a ser implementada deve possibilitar ao usuário informar o IP e a porta para conexões, não sendo esses campos estáticos. Neste trabalho, decidimos por implementar o jogo Batalha Naval.

Batalha Naval é um jogo de adivinhação em formato de tabuleiro/grade de dimensões $10x10$ para dois jogadores. Cada jogador possui um tabuleiro individual e 5 navios, cada navio possui um tamanho específico de valor 2 a 5, e estes devem ser alocados em algum local do tabuleiro de forma estritamente vertical ou horizontal, não podendo haver sobreposição entre eles [Carlos 1991].

O objetivo do jogo é afundar todos os navios do jogador inimigo antes que ele afunde todos os seus, ganhando aquele jogador que essa façanha primeiro. Cada jogador joga um turno de cada vez e vai tentando adivinhar, através de chamada de uma coordenada do tabuleiro, onde estão posicionados os navios no tabuleiro do adversário. Os estados de cada quadrante do tabuleiro podem ser os seguintes:

- $-$: $Vazio$. Não existe navio alocado;
- S: $Ocupado$. Existe uma parte do navio alocado;
- X: Hit. O jogador acertou a posição de um navio no tabuleiro;
- O: $Miss$. O jogador errou a posição de um navio no tabuleiro.

Neste trabalho, buscaremos demonstrar o nosso entendimento alcançado durante a atividade, descrevendo um pouco dos conceitos metodológicos aplicados como: sockets, protocolos de transporte e cliente/servidor, além dos detalhes mais importantes da solução de implementação encontrada e os resultados obtidos.

2. Metodologia e Visão Geral

A solução implementada utilizou-se dos conceitos de $socket$, linguagem Java, estruturação e implementação de arquitetura para comunicação cliente/servidor.

2.1. Cliente/Servidor

Este é o modelo mais utilizado para aplicações distribuídas em redes de computadores. No modelo cliente/servidor, a comunicação costuma se dar através de uma mensagem de solicitação do cliente enviada para o Servidor, pedindo para que alguma tarefa seja executada. Em seguida, o servidor executa a tarefa e envia a resposta [Tanenbaum 2003].

2.2. Socket

Patrício et al. (2015) descreve: "A tecnologia $socket$ é utilizada no desenvolvimento de aplicações do modelo cliente/servidor, para possibilitar a comunicação em rede. [Tanenbaum 2003] explica que cada socket tem um número que consiste no IP do $host$ mais um numero de 16 bits local para este $host$, denominado porta. Para que a comunicação funcione é necessário que uma conexão seja estabelecida entre um $socket$ da máquina transmissora e um $socket$ da máquina receptora"[Patricio 2015].

2.3. Linguagem Java para Aplicações em Rede

A linguagem Java auxilia bastante na implementação de programas que utilizam $sockets$. A classe $DatagramSocket$ gerencia um $socket$ UDP (não orientado à conexão), enquanto a classe $DatagramPacket$ gera um pacote de datagrama para ser enviado através do $socket$, ambas são elementos das bibliotecas nativas do Java [Oracle 2017]. Apesar de ser possível implementarmos o jogo utilizando protocolo de transporte TCP (orientado à conexão), por praticidade e por ser mais comumente utilizado em jogos, optamos por utilizar o protocolo de transporte UDP via datagramas.

Para enviar um pacote via $DatagramSocket$, é preciso definir um $DatagramPacket$ de saída informando os seguintes parâmetros: dados em um buffer (array de bytes), tamanho do pacote, endereço IP Inet e a porta destino, como demonstrado no exemplo abaixo retirado do código fonte do jogo:

```
1 DatagramSocket clienteSocket = new DatagramSocket() ;
2 clienteSocket.send(new DatagramPacket(
3         dados , dados.length , enderecoIpServidor , porta) ;
```

De forma semelhante, para receber um pacote via $DatagramSocket$, é preciso definir um $DatagramPacket$ de entrada informados os dados em um buffer (array de bytes), e o tamanho do pacote, como demonstrado no exemplo abaixo retirado do código fonte do jogo:

```
1 DatagramSocket clienteSocket = new DatagramSocket() ;
2 clienteSocket.send(new DatagramPacket(dados , dados.length) ;
```

3. Implementação

Foram criados dois subprojetos que compõem o projeto do jogo Batalha Naval: $BatalhaNaval_Cliente$ e $BaralhaNaval_servidor$. Descreveremos as principais classes que compõem cada um dos projetos e como é feita a comunicação entre elas.

3.1. Subprojeto BatalhaNavalCliente

Subprojeto que implementa todo o código necessário a ser executado no lado do cliente. As principais classes que compõem este subprojeto são:

1. AppCliente: Classe que implementa o método $main()$. Recupera informações de IP e porta do jogador e inicia jogo enviando as informações por parâmetro para a classe ClienteJogo
2. ClienteIO: Classe que é responsável por fazer a gestão de entrada e saída de dados via Socket Datagrama (UDP) pelo lado do cliente
3. ClienteJogo: Classe que administra o Jogo do lado do Cliente. Esta classe utiliza o ClienteIO para mandar e receber dados via Socket, armazena e apresenta as informações recebidas do servidor referentes ao jogo.

3.2. Subprojeto BatalhaNavalServidor

1. AppServer: Classe que implementa o método $main()$, pede informações de porta do servidor ao administrador e inicia jogo enviando as informações por parâmetro
2. ServidorIO: Classe que é responsável por fazer a gestão de entrada e saída de dados via Socket Datagrama (UDP) pelo lado do servidor
3. ServidorJogo: Classe que administra o Jogo do lado do servidor. Esta classe utiliza o ServidorIO para mandar e receber dados via Socket, a classe BatalhaNaval para persistir os dados do jogo, e armazena as seguintes informações de cada jogador: nome, ip, porta e se já alocou os navios
4. BatalhaNaval: Classe responsável por armazenar os tabuleiros do jogo de cada jogador. Define valores das posições, inicia tabuleiro, verifica jogada do usuário, atualiza tabuleiro, informa e processa estado do tabuleiro para a classe ServidorJogo. A estrutura de dados dos tabuleiros foi definida como duas matrizes de chars, cada uma referente ao tabuleiro de um jogador.

3.3. Comunicação entre o Cliente e o Servidor

Para comunicarmos entre uma máquina cliente e servidor, fizemos uso da técnica de Datagramas em linguagem Java, um processo já descrito anteriormente neste trabalho.

No lado do servidor, a classe AppServer configura a porta do servidor e inicia o jogo instanciando a classe ServidorJogo. A classe ServidorJogo por sua vez, inicia um loop que ouve e recebe pacotes do cliente, processa e devolve informações para o cliente utilizando os métodos da classe ServidorIO. Durante o processamento dos dados dentro do ServidorJogo, é feito chamadas à classe BatalhaNaval e tratamentos dos valores internos que permitem que as regras e dados do jogo sejam respeitadas e persistidas no servidor. O loop que envia e recebe pacotes do servidor foi implementado da seguinte forma:

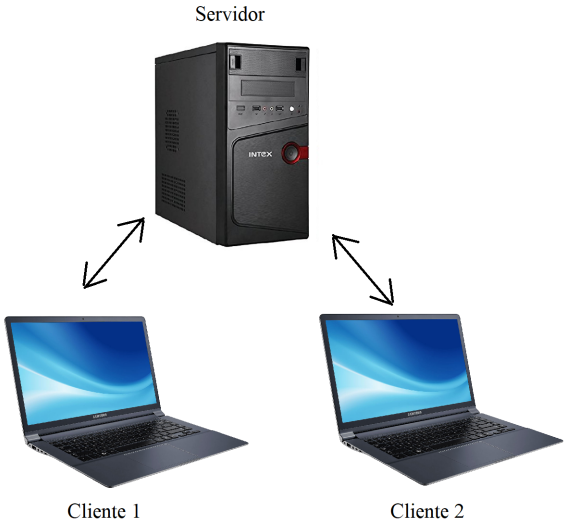

Figura 1. Representação da Comunicação Cliente/Servidor Implementada

```
1  ServidorIO io ← new ServidorIO() ;
2  procedimento start
3    enquanto (true) faca
4      informa "Aguardando recebimento de pacote..."
5      DatagramPacket pacote ← io.getPacote()
6      se pacote ≠ null entao
7        processaInput(pacote)
8        ...
9        io.enviaPacote(new DatagramPacket(
10         dados , tamanho , IP , porta))
11     fim se
12   fim enquanto
13 fim
```

No lado do cliente, também existe um loop que troca pacotes com o servidor. Antes do inicio do loop, é requerido do jogador que informe o IP e porta do servidor destino, além das posições onde serão alocados os navios no tabuleiro do usuário. Ao receber uma resposta positiva do servidor a respeito do recebimento das informações, o cliente passa a ficar no loop principal do jogo, onde o cliente enviará para o servidor a posição em que deseja acertar um navio no tabuleiro do inimigo e receberá de volta a situação dos tabuleiros e do jogo atualizados. O loop que envia e recebe pacotes do cliente foi implementado da seguinte forma:

```
1  ClienteIO io ← new ClienteIO() ;
2  procedimento iniciaJogo
3    enquanto (jogoAtivo) faca
4      io.enviaMensagem(usuario + ":" + pegaLinha()) ;
5      print ( Pacote enviado... Aguardando Replay... )
6      processaComando(io.getMensagem()) ;
7    fim enquanto
8  fim
```

4. Conclusão

Este relatório apresentou os resultados obtidos na realização da atividade da disciplina de Laboratório de Redes e consistiu no desenvolvimento de um jogo Batalha Naval que funciona em rede e utiliza $socket$ servidor e $socket$ cliente que enviam mensagens para controlar o jogo. A solução possibilita ao usuário informar o IP e a porta para conexões, não sendo esses campos estáticos. Também, descrevemos alguns conceitos aplicações Cliente/Servidor, uso de sockets e da linguagem Java para aplicações em rede. Com isso, é plausível afirmar que, houve uma considerável melhora em nossos conhecimentos da disciplina de Laboratório de Redes, principalmente na parte de implementação de aplicações em rede utilizando sockets.

Referências

Carlos (1991). *Rules for BattleShip*. Disponível em: https://www.cs.nmsu.edu/ bdu/-ta/487/brules.htm edition.

Oracle (2017). *Java, All About Datagram*.

Patricio (2015). *Jogo da Forca em Java ; Disponível em: http://periodicos.unesc.net/sulcomp/article/download/1005/941*.

Tanenbaum, A. S. (2003). *Sistemas Operacionais Modernos*. Prentice-Hall do Brasil, 2th edition.