Centro Federal de Educação Tecnológica de Minas Gerais
Campus VII - Unidade Timóteo - Engenharia da Computação
Trabalho Prático 1

Soquetes

Arthur Morais Pimentel

Trabalho prático de soquetes, implementação de um Jogo Da Velha usando protocolos TCP e UDP.

Professor: Lucas Pantuza Amorim

Timóteo, Setembro de 2022

1 Introdução

Neste trabalho foi implementado um programa que opera no modelo ponta-a-ponta que exercita a transmissão unidirecional ou a comunicação do tipo requisição-resposta sobre os protocolos UDP e TCP. Para fins didáticos, e consequentemente poder exercitar o contéudo teórico foi escolhido a funcionalidade de um jogo (Jogo da Velha), para poder contemplar a usabilidade da comunicação do tipo requisição-resposta sobre os protocolos estudados. Neste contexto foi aprofundando o uso de dois clientes, e um servidor que fará o intermedio entre eles.

2 Metodologia

Para desenvolvimento da aplicação foi utilizado a linguagem de programação Java, que disponibiliza de bibliotecas próprias para o desenvolvimento de comunicação na rede. Uma das mais importantes que foram de fundamental importância para o uso dos protocolos de comunicação foram DatagramPacket sit [2] juntamente com DatagramSocket sit [3], para a comunicação UDP, e ServerSocket sit [4] juntamente com Socket sit [5] para a comunicação TCP.

Foi criado uma interface atrávez da lib JFrame, própria da linguagem Java, onde a mesma proporciona uma melhor iteração do sistema com o usuário, e oferece uma melhor visibilidade das funcionalidades pautadas. De acordo com a figura capturada no inicio da execução do sistema, temos a seguinte imagem ilustrativa 1.

Figura 1: Interface gráfica

A figura 1, apresenta algumas funções a serem informadas pelo usuário, bem como o tabuleiro do jogo. O usuário precisa informar o tipo de conexão, sendo elas TCP e UDP, o tipo de aplicação, no caso sendo servidor ou cliente, para servidor a tela ficará inoperante, uma vez que o servidor atua apenas como uma ponte para comunicação dos clientes. Sendo optato um tipo de aplicação como cliente, o usuário precisa informar um nome como jogador, e informar o ip que o servidor está usando para assim poder realizar a conexão e iniciar o jogo. O jogo será iniciado apenas quando o servidor receber a conexão de dois clientes, após isso ele emitirá mensagens de inicio do game, o primeiro a iniciar a jogar será o primeiro que conectou ao servidor.

A IDE utilizada foi o IntelliJIDEA, o principal motivo foi poder subir as 3 aplicações ao mesmo tempo, de um mesmo projeto. Para isso foi necessário ativar a seguinte configuração que permitiu executar mais de uma aplicação ao mesmo tempo, sendo ela: "Allow multiple intances", ou pela tecla de atalho "Alt + U", que se encontra na tela "Run/Debug Configurations"da IDE. Após ativar a configuração, basta apenas iniciar 3 aplições, uma operando como servidor, e as duas restantes como clientes.
As conexões precisam ser do mesmo tipo para poder realizar as comunicações e poder efetuar as jogadas. Com o fim do jogo, ambas as 3 aplicações se encerram.

3 Resultados

Inicilizando o jogo como servidor, para poder receber a conexão dos clientes, ao clicar em conectar aparecerá uma mensagem informando que o servidor foi inicializado, conforme a figura 2

Figura 2: Inicialização do servidor

Após subir o servidor, na tela do mesmo não acontecerá mais nenhuma iteração, portanto agora é preciso subir os dois clientes, conforme a figura 3

(a) (b)

Figura 3: Clientes

Recebendo as mensagens de conectado, o jogo é inicializado, e com isso um cliente por vez vem a clicar em algum campo disponível do tabulereiro. Há um controle de usabilidade no que diz respeito quando é a vez de um cliente jogar, o outro então não consegue iterar com a tela, o servidor fica no aguarde da mensagem do cliente operante para encaminha-lá para o cliente ocioso, e assim procede o decorrer do jogo. Desta forma o fluxo vai procedindo até que o tabuleiro fique completo e dê velha, ou até que algum jogador ganhe. Conforme podemos visualizar nas figura 4 e 5 .

Figura 4: Vitória de um jogador

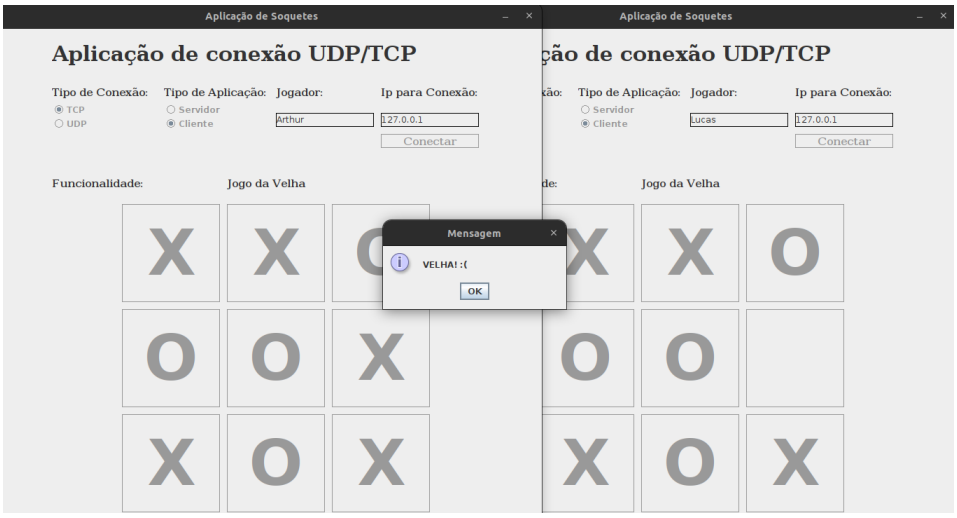

Figura 5: Derrota dos jogadores

4 Análise

Em meios de análise, pode-se observar como os dados são trafegados, dados dois tipos possíveis de conexão para o trabalho proposto, sendo elas a TCP e UDP, podemos ver mais sobre elas em sit [1].

Resumidamente temos que o protocolo UDP (User Datagram Protocol) tem como característica essencial, um atributo que é a falta de confiabilidade. Isso significa que, através da utilização desse protocolo, pode-se enviar datagramas de uma máquina à outra, mas sem garantia de que os dados enviados chegarão intactos e na ordem correta. Além do mais, o UDP é um protocolo que não é voltado à conexão. Isso significa que o "aperto de mão", ou, tecnicamente, handshake, não é necessário para que se estabeleça uma comunicação.

Diferente do UDP, o TCP é voltado à conexão e tem como garantia a integridade e ordem de todos os dados. Com o TCP, de fato temos uma conexão entre um ponto e outro, comumente chamados de servidor e cliente. É interessante notar que o TCP permite o envio simultâneo de dados de ambos os pontos ao outro, durante todo o fluxo de comunicação.

Vale frisar que para o trabalho proposto, em que é executado três aplicações, sendo elas servidor e clientes, temos que tanto o protocolo UDP quanto o protocolo TCP satisfarão a proposta inicial, que é efetuar uma comunicação entre diferentes aplicações, uma vez que as mesmas estarão sendo executada em localhost, ou seja, na máquina local. Sendo assim não se ver a necessidade que se teria para tratar algumas das possíveis falhas que apresenta o protocolo UDP.
Dito isso vemos pelas figuras 6 e 7, a captura das mensagens trafegadas tanto em UDP quanto em TCP. O software utilizado que é capaz de realizar estas capturas foi o Wireshark sit [6], que é um programa que analisa o tráfego de rede, e o organiza por protocolos.

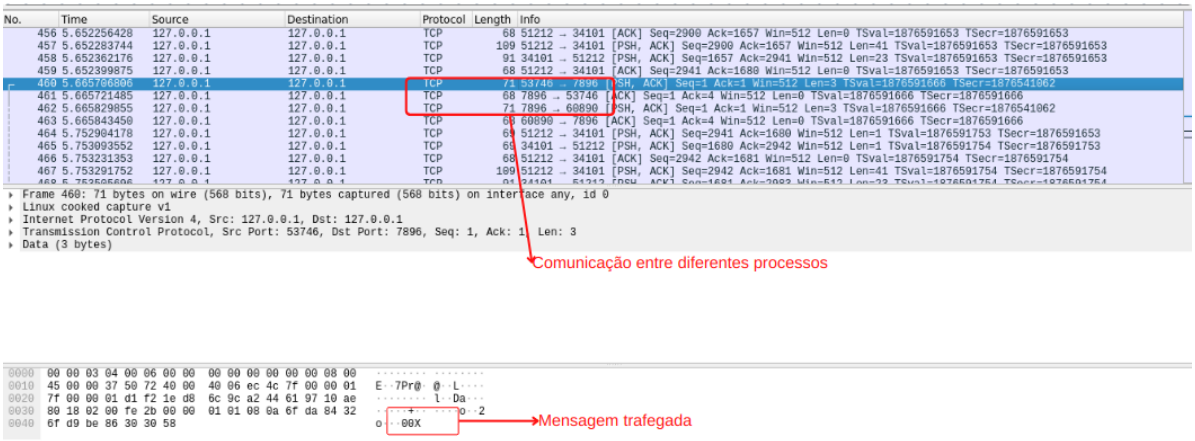

Figura 6: Comunicação TCP

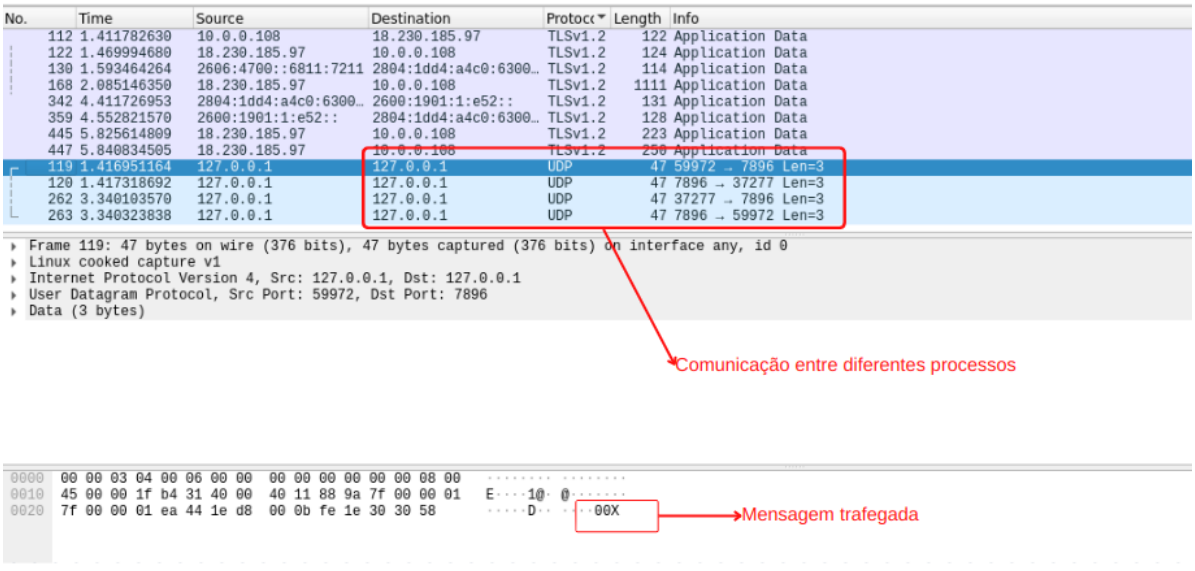

Figura 7: Comunicação UDP

5 Conclusão

Pode-se concluir que para a aplicação local o UDP realizou a comunicação tão bem quanto o TCP, dado a funcionalidade implementada. Até mesmo em uma comunicação entre duas máquinas que operam na mesma rede, o UDP mostrou realizar bem a comunicação.

Oque não seria trivial em caso de termos uma comunicação baseada em um cenário maior, onde teriamos mais dados a serem trafegados, e distintos endereços de rede, sendo necessário então realizar algumas tratativas para o protocolo UDP.

Com a realização do trabalho apresentado, foi possível entender na prática um pouco melhor sobre os protocolos de transporte estudados em sala de aula, bem como ter a experiência de capturar os dados trafegados em rede pelo Wireshark.

6 Código

Todo o código fonte se encontra disponível em GitHub.

Segue o código comentado:

Main - onde será inicializado a aplicação

```
1  package Main;
2
3  import Interface.Aplication;
4  import TCP.ClientTCP;
5  import TCP.ServerTCP;
6  import UDP.ClientUDP;
7  import UDP.ServerUDP;
8  import Utils.Window;
9
10
11 public  class Main implements Aplication {
12
13     /*
14     * Variaveis Estaticas
15     * */
16     public static Window window;
17
18     public static ClientTCP tcpClient;
19
20     public static ServerTCP tcpServer;
21
22     public static ClientUDP udpClient;
23
24     public static ServerUDP udpServer;
```

```
25
26      /*
27      * Inicializa a aplicacao,
28      * e a Jframe nomeada de Window
29      */
30      public static void main(String[] args) {
31          window = new Window();
32      }
33
34      /*
35      * Call back
36      * Quando clicado em conectar na Jframe
37      * ser direcionado para esta funcao
38      */
39      @Override
40      public void initialize() {
41          initializeAplication(window);
42      }
43      @Override
44      public void sendClient(){
45          if(tcpClient != null){
46              tcpClient.send();
47          }else{
48              if(udpClient != null){
49                  udpClient.send();
50              }
51          }
52
53      }
54
55
56
57      /*
58       *
59       * Conexao UDP
60       */
61      private void udpClient(Window window) {
62          udpClient = new ClientUDP(window);
63          udpClient.initializeUDPClient();
64      }
65
66      private void udpServer() {
67         udpServer = new ServerUDP();
68         udpServer.initializeUDPServer();
69      }
70
71      /*
72       *
```

```
73          * Conexao TCP
74          */
75         private void tcpClient(Window window) {
76             // arguments supply message and hostname
77             tcpClient = new ClientTCP(window);
78             tcpClient.initializeTCPClient();
79         }
80 
81         private void tcpServer() {
82             tcpServer = new ServerTCP();
83             tcpServer.initializeTCPServer();
84         }
85 
86 
87 
88         /*
89          *
90          * Inicializa Aplicacao
91          * com base nas configuracoes informadas pelo usuario na Jfram
92          */
93         private  void initializeAplication(Window window) {
94             if (window.getOptionConnection() == "TCP") {
95                 if (window.getAplicationConnection() == "Server") {
96                     window.isServer();
97                     tcpServer();
98                 } else {
99                     if (window.getAplicationConnection() == "Client") {
100                        tcpClient(window);
101                    }
102                }
103            } else {
104                if (window.getOptionConnection() == "UDP") {
105                    if (window.getAplicationConnection() == "Server") {
106                        window.isServer();
107                        udpServer();
108                    } else {
109                        if (window.getAplicationConnection() == "Client") {
110                            udpClient(window);
111                        }
112                    }
113                }
114            }
115        }
116 
117 }
```

codes/Main.java

TCP:

Servidor TCP

```java
package TCP;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

/*
* Servidor TCP
*/
public class ServerTCP {

    private final int serverPort = 7896;

    /*
    * Lista de Clientes que realizaram conexao com o servidor
    */
    private  List<Socket> clientsTCP = new ArrayList<Socket>();

    public void initializeTCPServer(){
        try {
            // Inicializacao do servidor
            int serverPort = 7896; // the server port
            ServerSocket listenSocket = new ServerSocket(serverPort);
            System.out.println("Server TCP Initialized");

            /*
            * Conexao
            * Servidor receber 2 clientes
            */
            while (clientsTCP.size() < 2) {
                // Aceita novo cliente
                Socket clientSocket = listenSocket.accept();
                if (!containsClient(clientSocket)) {
                    clientsTCP.add(clientSocket);
                }
            }

            /*
            * Apos ter sido realizada a conexao com os 2 clientes
            * E inicializado a comunicacao de fato
            */
            ComunicationTCP comunication = new ComunicationTCP(clientsTCP)
```

```
                ;

        } catch (IOException e) {
            System.out.println("Listen socket:" + e.getMessage());
        }
    }


    /*
    * Analisa se o cliente ja se conectou
    * Ou seja, um cliente pode efetuar apenas uma conexao
    */
    private boolean containsClient(Socket clientSocket) {
        boolean contains = false;
        String addressExisting;
        String address = String.valueOf(clientSocket.getInetAddress()) +"/"
            "+ String.valueOf(clientSocket.getPort());
        for (Socket client : clientsTCP) {
            addressExisting = String.valueOf(client.getInetAddress()) +"/"
                + String.valueOf(client.getPort());
            if (addressExisting.equals(address)) {
                contains = true;
                break;
            }
        }

        return contains;
    }

}
```

codes/TCP/ServerTCP.java

Cliente TCP

```
package TCP;

import Interface.InOut;
import Utils.Messages;
import Utils.Window;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

```

```java
/*
* Cliente TCP
*/
public class ClientTCP implements InOut {

    public    DataInputStream in;
    public    DataOutputStream out;
    public   Socket s;

    public   Window window;

    public ClientTCP(Window windowParam){
        window = windowParam;
    }

    /*
     * Funcao que efetua o inicio da comunicacao do Cliente
     */
    public void initializeTCPClient(){
        String host = window.getIpConnection();
        try {
            int serverPort = 7896;
            //Connection
            s = new Socket(host, serverPort);
            in = new DataInputStream(s.getInputStream());
            out = new DataOutputStream(s.getOutputStream());

            // mensagem de inicializacao enviado pelo servidor
            // esta mensagem inicial diferencia o tipo de cliente
            byte message = in.readByte();

            if (message == Messages.CONNECTCLIENT1.getValue()) {
                window.isClient("1");
                window.enableTable();
            } else {
                if (message == Messages.CONNECTCLIENT2.getValue()) {
                    window.isClient("2");
                    receive();
                }
            }

        } catch (UnknownHostException e) {
            System.out.println("Socket:" + e.getMessage());
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("readline:" + e.getMessage());
        }
```

```
62      }
63
64      /*
65      * Funcao que efetua o recebimento de mensagem
66      */
67      @Override
68      public  void receive() {
69          new Thread(){
70
71              @Override
72              public void run() {
73                  try {
74                      byte[] message = new byte[3];
75                       in.read(message);
76                      if(message[0] == Messages.FINISH.getValue()){
77                          s.close();
78                          System.exit(0);
79                      }else {
80                          window.updateTable(new String(message, "UTF-8"));
81                      }
82                  } catch (IOException e) {
83                      System.out.println("readline:" + e.getMessage());
84                  }
85              }
86          }.start();
87
88      }
89
90      /*
91       * Funcao que efetua o envio de mensagem
92       */
93      @Override
94      public  void send() {
95          try {
96              window.enableTable();
97              byte[] message = window.getPlayed().getBytes();
98              out.write(message);
99              window.resetPlayed();
100             window.disabledTable();
101             receive();
102         } catch (IOException e) {
103             System.out.println("readline:" + e.getMessage());
104         }
105     }
106
107 }
```

codes/TCP/ClientTCP.java

Comunicação TCP

```java
package TCP;

import Utils.Messages;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.*;

/*
* Classe onde a logica do jogo e executada
*/
public class ComunicationTCP {
    DataInputStream inClient1;
    DataOutputStream outClient1;

    DataInputStream inClient2;
    DataOutputStream outClient2;
    Socket clientSocket1;
    Socket clientSocket2;

    public ComunicationTCP(List<Socket> clients) {
      clientSocket1 = clients.get(0);
      clientSocket2 = clients.get(1);
      comunicationClients();
    }

    private void comunicationClients(){
        try {
            inClient1 = new DataInputStream(clientSocket1.getInputStream()
                );
            outClient1 = new DataOutputStream(clientSocket1.
                getOutputStream());

            inClient2 = new DataInputStream(clientSocket2.getInputStream()
                );
            outClient2 = new DataOutputStream(clientSocket2.
                getOutputStream());

            // Servidor enviar mensagem informando aos clientes a ordem de
                 prioridade
            // e que o jogo iniciou
            outClient1.writeByte(Messages.CONNECTCLIENT1.getValue());
            outClient2.writeByte(Messages.CONNECTCLIENT2.getValue());

```

```
42              byte[] message = new byte[3];
43
44              while(true){
45                  inClient1.read(message);
46                  String value = String.valueOf((char)message[0]);
47                  String finish = Messages.FINISH.getValueString();
48                  if(value.equals(finish)) break;
49
50                  outClient2.write(message);
51
52                  inClient2.read(message);
53                  value = String.valueOf((char)message[0]);
54                  if(value.equals(finish)) break;
55
56                  outClient1.write(message);
57              }
58
59              outClient1.writeByte(Messages.FINISH.getValue());
60              outClient2.writeByte(Messages.FINISH.getValue());
61              System.exit(0);
62
63          } catch (IOException e) {
64              System.out.println("Connection:" + e.getMessage());
65          } finally {
66              try {
67                  clientSocket1.close();
68                  clientSocket2.close();
69              } catch (IOException e) {/*close failed*/}
70          }
71
72      }
73
74  }
```

codes/TCP/ComunicationTCP.java

UDP:

Servidor UDP

```
1  package UDP;
2
3  import java.io.IOException;
4  import java.net.DatagramPacket;
5  import java.net.DatagramSocket;
6  import java.util.ArrayList;
7  import java.util.List;
```

```java

/*
 * Servidor UDP
 */
public class ServerUDP {

    private final int serverPort = 7896;

    private DatagramSocket aSocket = null;

    private List<DatagramPacket> clientsUDP = new ArrayList<DatagramPacket
        >();

    public void initializeUDPServer() {
        try {
            aSocket = new DatagramSocket(serverPort);
            System.out.println("Server UDP Initialized");
            byte[] message = new byte[1];
            /*
             * Conexao
             * Servidor receber 2 clientes
             */
            while (clientsUDP.size() < 2) {
                // Aceita novo cliente
                DatagramPacket request = new DatagramPacket(message,
                    message.length);
                aSocket.receive(request);

                if (!containsClient(request)) {
                    clientsUDP.add(request);
                }
            }

            /*
             * Apos ter sido realizada a conexao com os 2 clientes
             * E inicializado a comunicacao de fato
             */
            ComunicationUDP comunication = new ComunicationUDP(clientsUDP,
                aSocket);

        } catch (IOException e) {
            System.out.println("Listen socket:" + e.getMessage());
        }
    }


    /*
     * Analisa se o cliente ja se conectou
```

```
53          * Ou seja, um cliente pode efetuar apenas uma conexao
54          */
55         private boolean containsClient(DatagramPacket clientDatagram) {
56             boolean contains = false;
57             String addressExisting;
58             String address = String.valueOf(clientDatagram.getAddress()) + "/"
                   + String.valueOf(clientDatagram.getPort());
59             for (DatagramPacket client : clientsUDP) {
60                 addressExisting = String.valueOf(client.getAddress()) + "/" +
                       String.valueOf(client.getPort());
61                 if (addressExisting.equals(address)) {
62                     contains = true;
63                     break;
64                 }
65             }
66
67             return contains;
68         }
69
70 }
```

codes/UDP/ServerUDP.java

Cliente UDP

```
1  package UDP;
2
3  import Interface.InOut;
4  import Utils.Messages;
5  import Utils.Window;
6
7  import java.io.EOFException;
8  import java.io.IOException;
9  import java.net.*;
10
11 /*
12 * Cliente UDP
13 */
14 public class ClientUDP implements InOut {
15     public  Window window;
16
17     DatagramSocket aSocket = null;
18
19     InetAddress aHost;
20
21     int serverPort = 7896;
22
23     public ClientUDP(Window windowParam){
```

```
24          window = windowParam;
25      }
26
27      /*
28       * Funcao que efetua o inicio da comunicacao do Cliente
29       */
30      public void initializeUDPClient(){
31          String host = window.getIpConnection();
32          try {
33              //Connection
34              aSocket = new DatagramSocket();
35              aHost = InetAddress.getByName(host);
36
37              // Envia mensagem ao servidor
38              // Mensagem informando conecao
39              byte[] message = {Messages.CONNECTSERVER.getValue()};
40
41              DatagramPacket request =
42                      new DatagramPacket(message, message.length, aHost,
                            serverPort);
43              aSocket.send(request);
44
45              // Aguarda mensagem do servidor
46              DatagramPacket reply = new DatagramPacket(message, message.
                    length);
47              aSocket.receive(reply);
48
49              // Mensagem inicial diferencia o tipo de cliente
50              if (message[0] == Messages.CONNECTCLIENT1.getValue()) {
51                  window.isClient("1");
52                  window.enableTable();
53              } else {
54                  if (message[0] == Messages.CONNECTCLIENT2.getValue()) {
55                      window.isClient("2");
56                      receive();
57                  }
58              }
59
60          } catch (UnknownHostException e) {
61              System.out.println("Socket:" + e.getMessage());
62          } catch (EOFException e) {
63              System.out.println("EOF:" + e.getMessage());
64          } catch (IOException e) {
65              System.out.println("readline:" + e.getMessage());
66          }
67      }
68
69      /*
```

```
70      * Funcao que efetua o recebimento de mensagem
71      */
72     @Override
73     public  void receive() {
74         new Thread(){
75
76             @Override
77             public void run() {
78                 try {
79                     byte[] message = new byte[3];
80                     DatagramPacket reply = new DatagramPacket(message,
                           message.length);
81                     aSocket.receive(reply);
82                     if(message[0] == Messages.FINISH.getValue()){
83                         aSocket.close();
84                         System.exit(0);
85                     }else {
86                         window.updateTable(new String(message, "UTF-8"));
87                     }
88                 } catch (IOException e) {
89                     System.out.println("readline:" + e.getMessage());
90                 }
91             }
92         }.start();
93
94     }
95
96     /*
97      * Funcao que efetua o envio de mensagem
98      */
99     @Override
100    public  void send() {
101        try {
102            window.enableTable();
103            byte[] message = window.getPlayed().getBytes();
104            DatagramPacket request = new DatagramPacket(message, message.
                   length, aHost, serverPort);
105            aSocket.send(request);
106            window.resetPlayed();
107            window.disabledTable();
108            receive();
109        } catch (IOException e) {
110            System.out.println("readline:" + e.getMessage());
111        }
112    }
113
114 }
```

codes/UDP/ClientUDP.java

Comunicação UDP

```java
package UDP;

import Utils.Messages;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.List;

/*
 * Classe onde a logica do jogo e executada
 */
public class ComunicationUDP {

    private DatagramSocket aSocket = null;
    DatagramPacket clientSocket1;
    DatagramPacket clientSocket2;

    public ComunicationUDP(List<DatagramPacket> clients, DatagramSocket
        aSocket) {
      clientSocket1 = clients.get(0);
      clientSocket2 = clients.get(1);
      this.aSocket = aSocket;
      comunicationClients();
    }

    private void comunicationClients(){
        try {

            // Servidor enviar mensagem informando aos clientes a ordem de
                prioridade
            // e que o jogo iniciou

            byte[] messageConection = new byte[]{Messages.CONNECTCLIENT1.
                getValue()};

            DatagramPacket request =
                    new DatagramPacket(messageConection, messageConection.
                        length, clientSocket1.getAddress(), clientSocket1.
                        getPort());
            aSocket.send(request);

            messageConection = new byte[]{Messages.CONNECTCLIENT2.getValue
                ()};

            request =
```

```
41                  new DatagramPacket(messageConection, messageConection.
                        length, clientSocket2.getAddress(), clientSocket2.
                        getPort());
42              aSocket.send(request);
43
44              byte[] message = new byte[3];
45
46              while(true){
47                  DatagramPacket reply = new DatagramPacket(message, message
                        .length);
48                  aSocket.receive(reply);
49
50                  String value = String.valueOf((char)message[0]);
51                  String finish = Messages.FINISH.getValueString();
52                  if(value.equals(finish)) break;
53
54
55                  request =
56                          new DatagramPacket(message, message.length,
                                clientSocket2.getAddress(), clientSocket2.
                                getPort());
57                  aSocket.send(request);
58
59                  reply = new DatagramPacket(message, message.length);
60                  aSocket.receive(reply);
61                  value = String.valueOf((char)message[0]);
62                  if(value.equals(finish)) break;
63
64                  request =
65                          new DatagramPacket(message, message.length,
                                clientSocket1.getAddress(), clientSocket1.
                                getPort());
66                  aSocket.send(request);
67              }
68
69              byte[] messageFinish = new byte[]{Messages.FINISH.getValue()};
70
71              request =
72                      new DatagramPacket(messageFinish, messageFinish.length
                            , clientSocket1.getAddress(), clientSocket1.getPort
                            ());
73              aSocket.send(request);
74
75              request =
76                      new DatagramPacket(messageFinish, messageFinish.length
                            , clientSocket2.getAddress(), clientSocket2.getPort
                            ());
77              aSocket.send(request);
```

```
78
79              System.exit(0);
80
81          } catch (IOException e) {
82              System.out.println("Connection:" + e.getMessage());
83          }
84      }
85
86  }
```

codes/UDP/ComunicationUDP.java

Interfaces:

Estas interfaces implementadas para uso de callback, são utilizadas para dado certa iteração do usuário com a JFrame a mesma poder iteragir com outras classes.

Aplication

Usada principalmente na inicialização do fluxo de comunicação, apartir da tela da JFrame

```
1  package Interface;
2
3  public interface Aplication {
4      void initialize();
5
6      void sendClient();
7  }
```

codes/Interfaces/Aplication.java

InOut

Usada principalmente no envio e recebimento de mensagens por parte dos cliente UDP e TCP.

```
1  package Interface;
2
3  public interface InOut {
4
5      void send();
6
7      void receive();
8  }
```

codes/Interfaces/InOut.java

Utilitários

Messages

Trata-se de um Enum - Java, que abstrai algumas mensagens padrão para comunicação de ambos clientes e servidores.

```java
package Utils;

public enum Messages {

    CONNECTCLIENT1((byte)1),
    CONNECTCLIENT2((byte)2),

    CONNECTSERVER((byte) 3),
    FINISH((byte)4);


    private final byte value;

    Messages(byte value){
        this.value = value;
    }

    public byte getValue(){
        return value;
    }

    public String getValueString() {
        return String.valueOf(value);
    }

}
```

codes/Utils/Messages.java

Window

Por fim a classe responsável por criação e configuração da JFrame, na mesma se encontra toda a lógica e funcionalidade da janela iterativa java.

```java
package Utils;

import Interface.Aplication;
import Main.Main;

```

```
6  import javax.swing.*;
7  import java.awt.*;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.ActionListener;
10 import java.util.Objects;
11
12 public class Window  {
13
14     /*
15      * Variables
16      * */
17     JFrame jFrame = new JFrame();
18     Font big = new Font("Serif", Font.BOLD, 30);
19     Font small = new Font("Serif", Font.BOLD, 15);
20     Font button = new Font("Tahoma", Font.BOLD, 90);
21
22     String[][] table = new String[3][3];
23
24     String player;
25     String played = "";
26
27     // Type Connection
28     JRadioButton optionTCP;
29     JRadioButton optionUDP;
30
31     // Type Aplication
32     JRadioButton optionServer;
33     JRadioButton optionClient;
34
35     // Connection IP
36     JButton connectionAplication;
37
38     // Gamer
39
40     JLabel gamer;
41     JTextField nameGamer;
42
43     // Address ip
44     JTextField ip;
45
46     // Table buttons
47     JButton btn1;
48     JButton btn2;
49     JButton btn3;
50     JButton btn4;
51     JButton btn5;
52     JButton btn6;
53     JButton btn7;
```

```
54      JButton btn8;
55      JButton btn9;
56
57
58      public Window() {
59          initComponents();
60      }
61
62      private void initComponents() {
63          setConfigsWindow();
64          // Title
65          JLabel header = new JLabel("Aplica  o de conex o UDP/TCP");
66          jFrame.add(header);
67          header.setFont(big);
68          header.setBounds(100, 10, 600, 50);
69
70          // Conexion
71          JLabel typeConnection = new JLabel("Tipo de Conex o:");
72          jFrame.add(typeConnection);
73          typeConnection.setFont(small);
74          typeConnection.setBounds(100, 80, 150, 20);
75
76
77          ButtonGroup groupTypeConnection = new ButtonGroup();
78
79          optionTCP = new JRadioButton("TCP");
80          jFrame.add(optionTCP);
81          optionTCP.setBounds(100, 105, 150, 20);
82
83
84          optionUDP = new JRadioButton("UDP");
85          jFrame.add(optionUDP);
86          optionUDP.setBounds(100, 125, 150, 20);
87
88
89          groupTypeConnection.add(optionTCP);
90          groupTypeConnection.add(optionUDP);
91
92          // Aplication
93          JLabel typeApplication = new JLabel("Tipo de Aplica  o:");
94          jFrame.add(typeApplication);
95          typeApplication.setFont(small);
96          typeApplication.setBounds(260, 80, 150, 20);
97
98
99          ButtonGroup groupTypeApplication = new ButtonGroup();
100
101         optionServer = new JRadioButton("Servidor");
```

```
102         jFrame.add(optionServer);
103         optionServer.setBounds(260, 105, 150, 20);
104 
105 
106         optionClient = new JRadioButton("Cliente");
107         jFrame.add(optionClient);
108         optionClient.setBounds(260, 125, 150, 20);
109 
110 
111         groupTypeApplication.add(optionServer);
112         groupTypeApplication.add(optionClient);
113 
114         // Gamer
115         gamer = new JLabel("Jogador:");
116         jFrame.add(gamer);
117         gamer.setFont(small);
118         gamer.setBounds(420, 80, 150, 20);
119 
120         nameGamer = new JTextField(30);
121         jFrame.add(nameGamer);
122         nameGamer.setBounds(420, 120, 140, 20);
123         nameGamer.setBorder(BorderFactory.createMatteBorder(1, 1, 1, 1,
                Color.BLACK));
124 
125         // Host Conexion
126         JLabel ipConnection = new JLabel("Ip para Conex o:");
127         jFrame.add(ipConnection);
128         ipConnection.setFont(small);
129         ipConnection.setBounds(570, 80, 150, 20);
130 
131         ip = new JTextField(30);
132         jFrame.add(ip);
133         ip.setBounds(570, 120, 140, 20);
134         ip.setBorder(BorderFactory.createMatteBorder(1, 1, 1, 1, Color.
                BLACK));
135 
136         connectionAplication = new JButton("Conectar");
137         jFrame.add(connectionAplication);
138         connectionAplication.setFont(small);
139         connectionAplication.setBounds(570, 150, 140, 20);
140         connectionAplication.addActionListener(new ActionListener() {
141             @Override
142             public void actionPerformed(ActionEvent actionEvent) {
143                 initializeAplication();
144             }
145         });
146 
147 
```

```
148         // Funcionality
149         JLabel functionality = new JLabel("Funcionalidade:");
150         jFrame.add(functionality);
151         functionality.setFont(small);
152         functionality.setBounds(100, 210, 150, 20);
153
154
155         JLabel typeFunctionality = new JLabel("Jogo da Velha");
156         jFrame.add(typeFunctionality);
157         typeFunctionality.setFont(small);
158         typeFunctionality.setBounds(350, 210, 150, 20);
159
160
161         // Buttons
162
163         btn1 = new javax.swing.JButton();
164         jFrame.add(btn1);
165         btn1.setFont(button);
166         btn1.setBounds(200, 250, 140, 140);
167         btn1.addActionListener(new ActionListener() {
168             @Override
169             public void actionPerformed(ActionEvent actionEvent) {
170                 btn1ActionPerformed();
171             }
172         });
173
174
175         btn2 = new javax.swing.JButton();
176         jFrame.add(btn2);
177         btn2.setFont(button);
178         btn2.setBounds(350, 250, 140, 140);
179         btn2.addActionListener(new ActionListener() {
180             @Override
181             public void actionPerformed(ActionEvent actionEvent) {
182                 btn2ActionPerformed();
183             }
184         });
185
186
187         btn3 = new javax.swing.JButton();
188         jFrame.add(btn3);
189         btn3.setFont(button);
190         btn3.setBounds(500, 250, 140, 140);
191         btn3.addActionListener(new ActionListener() {
192             @Override
193             public void actionPerformed(ActionEvent actionEvent) {
194                 btn3ActionPerformed();
195             }
```

```
196         });
197
198
199         btn4 = new javax.swing.JButton();
200         jFrame.add(btn4);
201         btn4.setFont(button);
202         btn4.setBounds(200, 400, 140, 140);
203         btn4.addActionListener(new ActionListener() {
204             @Override
205             public void actionPerformed(ActionEvent actionEvent) {
206                 btn4ActionPerformed();
207             }
208         });
209
210
211         btn5 = new javax.swing.JButton();
212         jFrame.add(btn5);
213         btn5.setFont(button);
214         btn5.setBounds(350, 400, 140, 140);
215         btn5.addActionListener(new ActionListener() {
216             @Override
217             public void actionPerformed(ActionEvent actionEvent) {
218                 btn5ActionPerformed();
219             }
220         });
221
222
223         btn6 = new javax.swing.JButton();
224         jFrame.add(btn6);
225         btn6.setFont(button);
226         btn6.setBounds(500, 400, 140, 140);
227         btn6.addActionListener(new ActionListener() {
228             @Override
229             public void actionPerformed(ActionEvent actionEvent) {
230                 btn6ActionPerformed();
231             }
232         });
233
234
235         btn7 = new javax.swing.JButton();
236         jFrame.add(btn7);
237         btn7.setFont(button);
238         btn7.setBounds(200, 550, 140, 140);
239         btn7.addActionListener(new ActionListener() {
240             @Override
241             public void actionPerformed(ActionEvent actionEvent) {
242                 btn7ActionPerformed();
243             }
```

```
244         });
245
246
247         btn8 = new javax.swing.JButton();
248         jFrame.add(btn8);
249         btn8.setFont(button);
250         btn8.setBounds(350, 550, 140, 140);
251         btn8.addActionListener(new ActionListener() {
252             @Override
253             public void actionPerformed(ActionEvent actionEvent) {
254                 btn8ActionPerformed();
255             }
256         });
257
258
259         btn9 = new javax.swing.JButton();
260         jFrame.add(btn9);
261         btn9.setFont(button);
262         btn9.setBounds(500, 550, 140, 140);
263         btn9.addActionListener(new ActionListener() {
264             @Override
265             public void actionPerformed(ActionEvent actionEvent) {
266                 btn9ActionPerformed();
267             }
268         });
269
270
271     }//
272
273     private void setConfigsWindow() {
274         jFrame.setTitle("Aplica   o de Soquetes");
275         jFrame.setSize(800, 800);
276         jFrame.setLocationRelativeTo(null);
277         jFrame.setResizable(false);
278         jFrame.setVisible(true);
279         jFrame.setLayout(null);
280         jFrame.setDefaultCloseOperation(javax.swing.WindowConstants.
                EXIT_ON_CLOSE);
281     }
282
283     private void btn1ActionPerformed() {
284
285         if (Objects.equals(player, "1")) {
286             btn1.setText("X");
287             btn1.setEnabled(false);
288             table[0][0] = "X";
289             played = "OOX";
290             verifyWinner();
```

```
291         } else {
292             if (Objects.equals(player, "2")) {
293                 btn1.setText("O");
294                 btn1.setEnabled(false);
295                 table[0][0] = "O";
296                 played = "00O";
297                 verifyWinner();
298             }
299         }
300     }
301 
302     private void btn2ActionPerformed() {
303 
304         if (Objects.equals(player, "1")) {
305             btn2.setText("X");
306             btn2.setEnabled(false);
307             table[0][1] = "X";
308             played = "01X";
309             verifyWinner();
310         } else {
311             if (Objects.equals(player, "2")) {
312                 btn2.setText("O");
313                 btn2.setEnabled(false);
314                 table[0][1] = "O";
315                 played = "01O";
316                 verifyWinner();
317             }
318         }
319     }
320 
321     private void btn3ActionPerformed() {
322 
323         if (Objects.equals(player, "1")) {
324             btn3.setText("X");
325             btn3.setEnabled(false);
326             table[0][2] = "X";
327             played = "02X";
328             verifyWinner();
329         } else {
330             if (Objects.equals(player, "2")) {
331                 btn3.setText("O");
332                 btn3.setEnabled(false);
333                 table[0][2] = "O";
334                 played = "02O";
335                 verifyWinner();
336             }
337         }
338     }
```

```
339
340     private void btn4ActionPerformed() {
341
342         if (Objects.equals(player, "1")) {
343             btn4.setText("X");
344             btn4.setEnabled(false);
345             table[1][0] = "X";
346             played = "10X";
347             verifyWinner();
348         } else {
349             if (Objects.equals(player, "2")) {
350                 btn4.setText("O");
351                 btn4.setEnabled(false);
352                 table[1][0] = "O";
353                 played = "10O";
354                 verifyWinner();
355             }
356         }
357     }
358
359     private void btn5ActionPerformed() {
360
361         if (Objects.equals(player, "1")) {
362             btn5.setText("X");
363             btn5.setEnabled(false);
364             table[1][1] = "X";
365             played = "11X";
366             verifyWinner();
367         } else {
368             if (Objects.equals(player, "2")) {
369                 btn5.setText("O");
370                 btn5.setEnabled(false);
371                 table[1][1] = "O";
372                 played = "11O";
373                 verifyWinner();
374             }
375         }
376     }
377
378     private void btn6ActionPerformed() {
379
380         if (Objects.equals(player, "1")) {
381             btn6.setText("X");
382             btn6.setEnabled(false);
383             table[1][2] = "X";
384             played = "12X";
385             verifyWinner();
386         } else {
```

```
387             if (Objects.equals(player, "2")) {
388                 btn6.setText("O");
389                 btn6.setEnabled(false);
390                 table[1][2] = "O";
391                 played = "12O";
392                 verifyWinner();
393             }
394         }
395     }
396 
397     private void btn7ActionPerformed() {
398 
399         if (Objects.equals(player, "1")) {
400             btn7.setText("X");
401             btn7.setEnabled(false);
402             table[2][0] = "X";
403             played = "20X";
404             verifyWinner();
405         } else {
406             if (Objects.equals(player, "2")) {
407                 btn7.setText("O");
408                 btn7.setEnabled(false);
409                 table[2][0] = "O";
410                 played = "20O";
411                 verifyWinner();
412             }
413         }
414     }
415 
416     private void btn8ActionPerformed() {
417 
418         if (Objects.equals(player, "1")) {
419             btn8.setText("X");
420             btn8.setEnabled(false);
421             table[2][1] = "X";
422             played = "21X";
423             verifyWinner();
424         } else {
425             if (Objects.equals(player, "2")) {
426                 btn8.setText("O");
427                 btn8.setEnabled(false);
428                 table[2][1] = "O";
429                 played = "21O";
430                 verifyWinner();
431             }
432         }
433     }
434 
```

```
435     private void btn9ActionPerformed() {
436
437         if (Objects.equals(player, "1")) {
438             btn9.setText("X");
439             btn9.setEnabled(false);
440             table[2][2] = "X";
441             played = "22X";
442             verifyWinner();
443         } else {
444             if (Objects.equals(player, "2")) {
445                 btn9.setText("O");
446                 btn9.setEnabled(false);
447                 table[2][2] = "O";
448                 played = "22O";
449                 verifyWinner();
450             }
451         }
452     }
453
454     private void initializeAplication() {
455         if (getOptionConnection() == null) {
456             alertMessage("   preciso informar um tipo de conex o!");
457         } else {
458             if (getAplicationConnection() == null) {
459                 alertMessage("   preciso informar um tipo de Aplica   o!"
                        );
460             } else {
461                 if (getAplicationConnection() == "Server") {
462                     setIpConnection("127.0.0.1");
463                     startAplication();
464                 } else {
465                     if (nameGamer.getText().isEmpty()) {
466                         alertMessage("   preciso informar um nome!");
467                     } else {
468                         if (ip.getText().isEmpty()) {
469                             alertMessage("   preciso informar um Ip!");
470                         } else {
471                             startAplication();
472                         }
473                     }
474                 }
475
476             }
477         }
478     }
479
480     private void sendMove(){
481         Aplication inOut = new Main();
```

```
482         inOut.sendClient();
483     }
484 
485     private void startAplication() {
486         connectionAplication.setEnabled(false);
487         Aplication aplication = new Main();
488         aplication.initialize();
489     }
490 
491     private void verifyWinner() {
492         //Check Lines
493         for (int i = 0; i < 3; i++) {
494             if (table[i][0] == "X" && table[i][1] == "X" && table[i][2] ==
                    "X") {
495                 JOptionPane.showMessageDialog(null, nameGamer.getText() +
                        " ganhador!!!");
496                 endOfTheGame();
497             }
498             if (table[i][0] == "O" && table[i][1] == "O" && table[i][2] ==
                    "O") {
499                 JOptionPane.showMessageDialog(null, nameGamer.getText() +
                        " ganhador!!!");
500                 endOfTheGame();
501             }
502         }
503 
504         //Check Columns
505         for (int i = 0; i < 3; i++) {
506             if (table[0][i] == "X" && table[1][i] == "X" && table[2][i] ==
                    "X") {
507                 JOptionPane.showMessageDialog(null, nameGamer.getText() +
                        " ganhador!!!");
508                 endOfTheGame();
509             }
510             if (table[0][i] == "O" && table[1][i] == "O" && table[2][i] ==
                    "O") {
511                 JOptionPane.showMessageDialog(null, nameGamer.getText() +
                        " ganhador!!!");
512                 endOfTheGame();
513             }
514         }
515 
516         //Check Main Diagonal
517         if (table[0][0] == "X" && table[1][1] == "X" && table[2][2] == "X"
                ) {
518             JOptionPane.showMessageDialog(null, nameGamer.getText() + "
                    ganhador!!!");
519             endOfTheGame();
```

```
520         }
521         if (table[0][0] == "O" && table[1][1] == "O" && table[2][2] == "O"
                ) {
522             JOptionPane.showMessageDialog(null, nameGamer.getText() + "
                    ganhador!!!");
523             endOfTheGame();
524         }
525
526         //Check Secondary Diagonal
527         if (table[0][2] == "X" && table[1][1] == "X" && table[2][0] == "X"
                ) {
528             JOptionPane.showMessageDialog(null, nameGamer.getText() + "
                    ganhador!!!");
529             endOfTheGame();
530         }
531         if (table[0][2] == "O" && table[1][1] == "O" && table[2][0] == "O"
                ) {
532             JOptionPane.showMessageDialog(null, nameGamer.getText() + "
                    ganhador!!!");
533             endOfTheGame();
534         }
535
536         verifyLoss();
537     }
538
539     private void verifyLoss() {
540         if(!played.equals(Messages.FINISH.getValueString())){
541             boolean loss = true;
542
543             for (int i = 0; i < 3; i++) {
544                 for (int j = 0; j < 3; j++) {
545                     if (table[i][j] == null) {
546                         loss = false;
547                     }
548                 }
549             }
550
551             if (loss) {
552                 JOptionPane.showMessageDialog(null, "VELHA! :(");
553                 endOfTheGame();
554             }
555         }
556
557         sendMove();
558     }
559
560     public void disabledTable() {
561         btn1.setEnabled(false);
```

```
562         btn2.setEnabled(false);
563         btn3.setEnabled(false);
564         btn4.setEnabled(false);
565         btn5.setEnabled(false);
566         btn6.setEnabled(false);
567         btn7.setEnabled(false);
568         btn8.setEnabled(false);
569         btn9.setEnabled(false);
570     }
571 
572     public void enableTable() {
573         btn1.setEnabled(true);
574         btn2.setEnabled(true);
575         btn3.setEnabled(true);
576         btn4.setEnabled(true);
577         btn5.setEnabled(true);
578         btn6.setEnabled(true);
579         btn7.setEnabled(true);
580         btn8.setEnabled(true);
581         btn9.setEnabled(true);
582     }
583 
584     public void updateTable(String coordinates) {
585         enableTable();
586         String[] values = coordinates.split("");
587         int x = Integer.parseInt(values[0]);
588         int y = Integer.parseInt(values[1]);
589         if (table[x][y] == null) {
590             table[x][y] = values[2];
591             updateTableButtons();
592         }
593     }
594     private void updateTableButtons() {
595         if (table[0][0] != null) {
596             btn1.setText(table[0][0]);
597             btn1.setEnabled(false);
598         }
599         if (table[0][1] != null) {
600             btn2.setText(table[0][1]);
601             btn2.setEnabled(false);
602         }
603         if (table[0][2] != null) {
604             btn3.setText(table[0][2]);
605             btn3.setEnabled(false);
606         }
607         if (table[1][0] != null) {
608             btn4.setText(table[1][0]);
609             btn4.setEnabled(false);
```

```
610             }
611             if (table[1][1] != null) {
612                 btn5.setText(table[1][1]);
613                 btn5.setEnabled(false);
614             }
615             if (table[1][2] != null) {
616                 btn6.setText(table[1][2]);
617                 btn6.setEnabled(false);
618             }
619             if (table[2][0] != null) {
620                 btn7.setText(table[2][0]);
621                 btn7.setEnabled(false);
622             }
623             if (table[2][1] != null) {
624                 btn8.setText(table[2][1]);
625                 btn8.setEnabled(false);
626             }
627             if (table[2][2] != null) {
628                 btn9.setText(table[2][2]);
629                 btn9.setEnabled(false);
630             }
631 
632         }
633 
634         private void setIpConnection(String address) {
635             ip.setText(address);
636             ip.setEditable(false);
637         }
638 
639         public String getIpConnection() {
640             return ip.getText();
641         }
642 
643         public void alertMessage(String message) {
644             JOptionPane.showMessageDialog(null, message);
645         }
646 
647         public String getOptionConnection() {
648             if (optionTCP.isSelected()) {
649                 return "TCP";
650             }
651             if (optionUDP.isSelected()) {
652                 return "UDP";
653             }
654             return null;
655         }
656 
657         public String getAplicationConnection() {
```

```
658             if (optionServer.isSelected()) {
659                 return "Server";
660             }
661             if (optionClient.isSelected()) {
662                 return "Client";
663             }
664             return null;
665         }
666 
667         public void endOfTheGame(){
668             played = Messages.FINISH.getValueString();
669         }
670         public void resetPlayed(){
671             played = "";
672         }
673 
674         public String getPlayed(){
675             return played;
676         }
677 
678 
679         public void isServer() {
680             disabledTable();
681             nameGamer.setText("Servidor");
682             nameGamer.setEditable(false);
683             disableOptionsConnections();
684             alertMessage("Servidor Inicializado");
685         }
686 
687         public void isClient(String idPlayer){
688             disabledTable();
689             nameGamer.setEditable(false);
690             player = idPlayer;
691             ip.setEditable(false);
692             disableOptionsConnections();
693             alertMessage(nameGamer.getText() + " conectado ao servidor!");
694         }
695         private void disableOptionsConnections(){
696             optionClient.setEnabled(false);
697             optionServer.setEnabled(false);
698             optionTCP.setEnabled(false);
699             optionUDP.setEnabled(false);
700         }
701 
702 }
```

codes/Utils/Window.java

Referências

[1] *TCP/UDP*, . URL https://www.alura.com.br/artigos/quais-as-diferencas-entre-o-tcp-e-o-udp.

[2] *DatagramPacket*, . URL https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html.

[3] *DatagramSocket*, . URL https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html.

[4] *ServerSocket*, . URL https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html.

[5] *Socket*, . URL https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html.

[6] *Wireshark*, . URL https://www.wireshark.org/.