**Лабораторная работа 2**

**Java network game**

**A network Tic-Tac-Toe game**

Here is the server for multiple two-player games. It listens for two clients to connect, and spawns a thread for each: the first is Player X and the second is Player O. The client and server send simple string messages back and forth to each other; messages correspond to the Tic Tac Toe protocol, which I made up for this example.

|  |
| --- |
| import java.io.IOException;import java.io.PrintWriter;import java.net.ServerSocket;import java.net.Socket;import java.util.Arrays;import java.util.Scanner;import java.util.concurrent.Executors;/\*\* \* A server for a multi-player tic tac toe game. Loosely based on an example in \* Deitel and Deitel’s “Java How to Program” book. For this project I created a \* new application-level protocol called TTTP (for Tic Tac Toe Protocol), which \* is entirely plain text. The messages of TTTP are: \* \* Client -> Server \*     MOVE <n> \*     QUIT \* \* Server -> Client \*     WELCOME <char> \*     VALID\_MOVE \*     OTHER\_PLAYER\_MOVED <n> \*     OTHER\_PLAYER\_LEFT \*     VICTORY \*     DEFEAT \*     TIE \*     MESSAGE <text> \*/public class TicTacToeServer {    public static void main(String[] args) throws Exception {        try (var listener = new ServerSocket(58901)) {            System.out.println("Tic Tac Toe Server is Running...");            var pool = Executors.newFixedThreadPool(200);            while (true) {                Game game = new Game();                pool.execute(game.new Player(listener.accept(), 'X'));                pool.execute(game.new Player(listener.accept(), 'O'));            }        }    }}class Game {    // Board cells numbered 0-8, top to bottom, left to right; null if empty    private Player[] board = new Player[9];    Player currentPlayer;    public boolean hasWinner() {        return (board[0] != null && board[0] == board[1] && board[0] == board[2])            || (board[3] != null && board[3] == board[4] && board[3] == board[5])            || (board[6] != null && board[6] == board[7] && board[6] == board[8])            || (board[0] != null && board[0] == board[3] && board[0] == board[6])            || (board[1] != null && board[1] == board[4] && board[1] == board[7])            || (board[2] != null && board[2] == board[5] && board[2] == board[8])            || (board[0] != null && board[0] == board[4] && board[0] == board[8])            || (board[2] != null && board[2] == board[4] && board[2] == board[6]        );    }    public boolean boardFilledUp() {        return Arrays.stream(board).allMatch(p -> p != null);    }    public synchronized void move(int location, Player player) {        if (player != currentPlayer) {            throw new IllegalStateException("Not your turn");        } else if (player.opponent == null) {            throw new IllegalStateException("You don't have an opponent yet");        } else if (board[location] != null) {            throw new IllegalStateException("Cell already occupied");        }        board[location] = currentPlayer;        currentPlayer = currentPlayer.opponent;    }    /\*\*     \* A Player is identified by a character mark which is either 'X' or 'O'.     \* For communication with the client the player has a socket and associated     \* Scanner and PrintWriter.     \*/    class Player implements Runnable {        char mark;        Player opponent;        Socket socket;        Scanner input;        PrintWriter output;        public Player(Socket socket, char mark) {            this.socket = socket;            this.mark = mark;        }        @Override        public void run() {            try {                setup();                processCommands();            } catch (Exception e) {                e.printStackTrace();            } finally {                if (opponent != null && opponent.output != null) {                    opponent.output.println("OTHER\_PLAYER\_LEFT");                }                try {socket.close();} catch (IOException e) {}            }        }        private void setup() throws IOException {            input = new Scanner(socket.getInputStream());            output = new PrintWriter(socket.getOutputStream(), true);            output.println("WELCOME " + mark);            if (mark == 'X') {                currentPlayer = this;                output.println("MESSAGE Waiting for opponent to connect");            } else {                opponent = currentPlayer;                opponent.opponent = this;                opponent.output.println("MESSAGE Your move");            }        }        private void processCommands() {            while (input.hasNextLine()) {                var command = input.nextLine();                if (command.startsWith("QUIT")) {                    return;                } else if (command.startsWith("MOVE")) {                    processMoveCommand(Integer.parseInt(command.substring(5)));                }            }        }        private void processMoveCommand(int location) {            try {                move(location, this);                output.println("VALID\_MOVE");                opponent.output.println("OPPONENT\_MOVED " + location);                if (hasWinner()) {                    output.println("VICTORY");                    opponent.output.println("DEFEAT");                } else if (boardFilledUp()) {                    output.println("TIE");                    opponent.output.println("TIE");                }            } catch (IllegalStateException e) {                output.println("MESSAGE " + e.getMessage());            }        }    }} |

These days, games like this would be played with clients in a web browser, and the server would be a web server (likely using a WebSockets library). But today, we’re learning about programming directly with sockets, on custom ports, with custom protocols, so we’re sticking with Java for our custom clients. The first version of this program was written in about 2002, so it uses...wait for it...Java Swing!

|  |
| --- |
| import java.awt.Font;import java.awt.Color;import java.awt.GridLayout;import java.awt.GridBagLayout;import java.awt.BorderLayout;import java.awt.event.MouseAdapter;import java.awt.event.MouseEvent;import java.util.Scanner;import java.io.PrintWriter;import java.net.Socket;import javax.swing.JFrame;import javax.swing.JLabel;import javax.swing.JOptionPane;import javax.swing.JPanel;/\*\* \* A client for a multi-player tic tac toe game. Loosely based on an example in \* Deitel and Deitel’s “Java How to Program” book. For this project I created a \* new application-level protocol called TTTP (for Tic Tac Toe Protocol), which \* is entirely plain text. The messages of TTTP are: \* \* Client -> Server \*     MOVE <n> \*     QUIT \* \* Server -> Client \*     WELCOME <char> \*     VALID\_MOVE \*     OTHER\_PLAYER\_MOVED <n> \*     OTHER\_PLAYER\_LEFT \*     VICTORY \*     DEFEAT \*     TIE \*     MESSAGE <text> \*/public class TicTacToeClient {    private JFrame frame = new JFrame("Tic Tac Toe");    private JLabel messageLabel = new JLabel("...");    private Square[] board = new Square[9];    private Square currentSquare;    private Socket socket;    private Scanner in;    private PrintWriter out;    public TicTacToeClient(String serverAddress) throws Exception {        socket = new Socket(serverAddress, 58901);        in = new Scanner(socket.getInputStream());        out = new PrintWriter(socket.getOutputStream(), true);        messageLabel.setBackground(Color.lightGray);        frame.getContentPane().add(messageLabel, BorderLayout.SOUTH);        var boardPanel = new JPanel();        boardPanel.setBackground(Color.black);        boardPanel.setLayout(new GridLayout(3, 3, 2, 2));        for (var i = 0; i < board.length; i++) {            final int j = i;            board[i] = new Square();            board[i].addMouseListener(new MouseAdapter() {                public void mousePressed(MouseEvent e) {                    currentSquare = board[j];                    out.println("MOVE " + j);                }            });            boardPanel.add(board[i]);        }        frame.getContentPane().add(boardPanel, BorderLayout.CENTER);    }    /\*\*     \* The main thread of the client will listen for messages from the server.     \* The first message will be a "WELCOME" message in which we receive our     \* mark. Then we go into a loop listening for any of the other messages,     \* and handling each message appropriately. The "VICTORY", "DEFEAT", "TIE",     \*  and "OTHER\_PLAYER\_LEFT" messages will ask the user whether or not to     \* play another game. If the answer is no, the loop is exited and the server     \* is sent a "QUIT" message.     \*/    public void play() throws Exception {        try {            var response = in.nextLine();            var mark = response.charAt(8);            var opponentMark = mark == 'X' ? 'O' : 'X';            frame.setTitle("Tic Tac Toe: Player " + mark);            while (in.hasNextLine()) {                response = in.nextLine();                if (response.startsWith("VALID\_MOVE")) {                    messageLabel.setText("Valid move, please wait");                    currentSquare.setText(mark);                    currentSquare.repaint();                } else if (response.startsWith("OPPONENT\_MOVED")) {                    var loc = Integer.parseInt(response.substring(15));                    board[loc].setText(opponentMark);                    board[loc].repaint();                    messageLabel.setText("Opponent moved, your turn");                } else if (response.startsWith("MESSAGE")) {                    messageLabel.setText(response.substring(8));                } else if (response.startsWith("VICTORY")) {                    JOptionPane.showMessageDialog(frame, "Winner Winner");                    break;                } else if (response.startsWith("DEFEAT")) {                    JOptionPane.showMessageDialog(frame, "Sorry you lost");                    break;                } else if (response.startsWith("TIE")) {                    JOptionPane.showMessageDialog(frame, "Tie");                    break;                } else if (response.startsWith("OTHER\_PLAYER\_LEFT")) {                    JOptionPane.showMessageDialog(frame, "Other player left");                    break;                }            }            out.println("QUIT");        } catch (Exception e) {            e.printStackTrace();        }        finally {            socket.close();            frame.dispose();        }    }    static class Square extends JPanel {        JLabel label = new JLabel();        public Square() {            setBackground(Color.white);            setLayout(new GridBagLayout());            label.setFont(new Font("Arial", Font.BOLD, 40));            add(label);        }        public void setText(char text) {            label.setForeground(text == 'X' ? Color.BLUE : Color.RED);            label.setText(text + "");        }    }    public static void main(String[] args) throws Exception {        if (args.length != 1) {            System.err.println("Pass the server IP as the sole command line argument");            return;        }        TicTacToeClient client = new TicTacToeClient(args[0]);        client.frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);        client.frame.setSize(320, 320);        client.frame.setVisible(true);        client.frame.setResizable(false);        client.play();    }} |