Tic Tac Toe Network Protocol

1.0 Abstract

This document outlines the protocol specification for a networked Tic Tac Toe game. The game uses a client-server model to connect players, and supports simple matchmaking. It is currently not possible for two players to specifically request to connect to a game together - the server pairs the first two players to connect and spawns a game thread, then waits for another two players.

For my CS494 term project, this protocol is is implemented in Java. To keep implementation easy, the Swing library is used for interface design. Any socket program following the protocol should be able to implement client or server software.

2.0 Table of Contents

3.0 Introduction

3.1 Game Flow

A server process handles communication between two clients, passes moves between them, and is in charge of checking wins/losses every turn. The protocol was designed to support two clients per server, as Tic Tac Toe is a two-player game. The server begins, and listens for two TCP connections. Once two connections have been made, the server gets a player name from each connected client, and delegates the 'X' and 'O' symbols based on who connected first ('X' is player 1, 'O' is player 2). A loop then begins where clients take turns making moves until:

1. A player wins the game
2. A player voluntarily quits the game
3. The connection is broken

If the server calculates that a player has won, it sends an appropriate message to each of the clients updating them on the game state. The server then closes the TCP connection for each client and it is up to the client to detect the closing of the connection and handle accordingly. If the connection between the server and a client is broken, the server will send an appropriate message to the other client and close the connection.

3.2 Character Encoding

The Java implementation of this protocol expects UTF-8 character encoding for messages. However, all messages specified in the protocol consist of characters represented in UTF-8 in the decimal range 0-255, and could likely be parsed using an array of ASCII characters (this is untested, and based on conjecture).

3.3 Messages

The payload of each packet of data consists of a string of maximum size of 8 bytes encoded in UTF-8. It is up to the client and server to parse the string data into other datatypes as needed (e.g. converting to integers for the x and y coordinates of a chosen move). A buffer 8 bytes on both the client and server holds incoming messages. Messages sent to and from the server during the setup stage of the game may not be prefixed with a command, and may contain raw data. Once the game enters its main turn-based loop however, incoming data are expected to be preceded by a command message denoting what kind of data is coming next.

4.0 Message Details

4.1 Get setup info

Command: SETUPINFO
Sent Parameters: <player symbol> <other player name>
Received Parameters: <player name>

Sent from the server to both clients after they have connected. Expects a string in response containing the chosen nickname for the player. The server sends the delegated player symbol ('X' or 'O') to the player when a nickname has been received from the client. When a nickname has been received from each client, the server sends the other player's nickname to each of the client's.

4.2 Your move

Command: YOURMOVE
Sent Parameters: None
Received Parameters: <x> <y>

Sent from the server to a client to denote it is their turn to make a move. The server expects a string containing two colon-separated integers in response, which are the (x, y) coordinates of the player's chosen move. The server performs a check to verify a move has not already been made and will send a "bad move" message in response if the move is invalid.

4.3 Bad move

Command: FAILMOVE
Sent Parameters: None
Received Parameters: <x> <y>

Sent from the server to a client to denote the move last sent was invalid. The server expects a string of two colon-separated integers in response, which are the (x, y) coordinates of the player's chosen move. The server performs a check to verify a move has not already been made and will send another bad move in response if the move is invalid.

4.4 Other move

Command: OTHERMOV
Sent Parameters: <x> <y> <symbol> <winning move>
Received Parameters: None

Sent from the server to a client to denote the move made by the other player. The parameters of the message are sent as a colon-separated string. The first two parameters specify the x and y coordinates of the move. The third parameter specifies the symbol of the player who made the move. The fourth parameter

denotes whether a winning move was made. If the move was the move that won the game, the parameter will contain the symbol of the winning player. If the move did not win the game, the parameter will contain a ‘ ‘ (space) character.

4.5 My move

Command: MYMOVE!!
Sent Parameters: <x> <y>
Received Parameters: None

Sent from a client to the server to denote their chosen move for their turn. After sending the command the client will wait for an acknowledgement message from the server before sending parameters. Parameters are sent as a second string of colon-separated (x, y) coordinates of the move location chosen by the player.

4.6 You win

Command: YOUWIN!!
Sent Parameters: None
Received Parameters: None

Sent from the server to a client to denote they have won the game. The server will then close the connection to the client. The client is in charge of handling the closing of the connection, and relaying the win to the user.

4.7 You lose

Command: YOULOSE!
Send Parameters: None
Received Parameters: None

Sent from the server to a client to denote they have lost the game. The server will then close the connection to the client. The client is in charge of handling the closing of the connection, and relaying the loss to the user.

4.8 Quit now

Command: QUITNOW!
Send Parameters: None
Received Parameters: None

Can be sent either from the server or a client. This is a generic message to denote that the game is ending before it has been won. Sent from a client if they wish to end the game early. If the server receives this message from the client or a client unexpectedly disconnects, the message will be relayed to the other client and both connections will be closed.

4.9 Acknowledge

Command: ACKMESSG
Send Parameters: None
Received Parameters: None

Can be sent from either the server or a client. Used to acknowledge a command message that takes no parameters in response.