Tic Tac Toe Protocol

**Introduction**

This protocol is intended to provide the communication service needed to play a Tic Tac Toe game between two hosts, a client and a server. The game is meant to be played between a client and a server, with the server playing multiply clients at the same time. This document describes only the protocol for the messages exchanged over the network and does not explain how to use the program or what the user is expected to see.

**Protocol Specification**

All messages sent between the client and the server will be sent over UDP. The server will identify each client by their IP and port address. All messages will be encoded strings. All messages sent from the client to the server will be prepended with a unique ID, and the server will prepend the same ID to its reply. This is so that duplicated messages will not be processed twice, and that all messages will be processed in order. The ID should start from 0 and increment by 1 on each new message. For example, if the first message is “X”, then the encoded string “0 X” will be sent, and the response will be “0 0”.

If the server receives the same ID as the last request, it will re-send the last reply, with should be saved in memory. If the server receives an older ID, it will ignore the message. If the server receives a new ID, it will respond with a new reply.

In case the client does not get a respond before a timeout (set to 0.5 second) , it will retry sending the request for a maximum of 10 times before deciding that the server is not available and exiting the game.

If the server will be constantly sending “ping” to the client to check if the client is still active, to which the client will respond with “pong”. This is the only exchange between the server and the client that does not use an ID since it is initiated by the server. If the server does not get a “pong” for 5 “ping”s, one at each second, the server will exit the game.

During the game the client and the server will send their move to each other at their turn. The moves will be encoded 1,2,3,4,5,6,7,8,9, each representing a position on the tic tac toe board as such:

7 │ 8 │ 9

───┼───┼───

4 │ 5 │ 6

───┼───┼───

1 │ 2 │ 3

The client and the server will both keep their own state, but the server determine when the game ends.

The client will initiate the connection when the game starts by sending one of the two messages:

1. “X” if the client is to play the first move.
2. “O” if the server is to play the first move.

To which the server will respond with “<move> <status>”. Where <move> is the ([1-9]) server’s first move, or 0 if it does not move first. And <status> is an integer 0-3 that represents the status of the game as such:

0: In progress, 1: client has won, 2: server has won, 3: game has ended in a tie

For example, the server will reply with:

1. “0 0” to acknowledge the start of a new game with the client moving first.
2. Or, “5 0” if the server choose position 5 as the first move.

Next, the client will send its move “<move>” to the server. The move should be validated before sending to the server so that it will not contain anything other than 1 to 9 and it cannot be a move already used. To this the server will respond its next “<move> <status>”. This step will be repeated until <status> is anything other than 0.

If the status is anything other than 0, then the game has ended, and at this point the client will send “close” to the server and exit. The server will exit the session if it has received the “close”, or it will exit after 5 un-responded “ping”.

The protocol expects to work as intended with the proper setup. Messages other than the ones specified by this protocol may cause unexpected behavior. The program will try to catch errors without crashing, but it will not be able to guarantee coverage for all edge cases.

**Example**

This is an example of all the messages exchanges during a possible game, with the exception of “ping” and “pong”:

CLIENT: “0 X” id=0. The client request to start a game with it playing ‘X’.

SERVER: “0 0 0” id=0. The server responded with no move. Status is in progress.

CLIENT: “1 1” id=1. The client choose position 1

SERVER: “1 4 0” id=1. The server choose position 4. Status is in progress.

CLIENT: “2 5” id=2. The client choose position 5.

SERVER: “2 7 0” id=2. The server choose position 7. Status is in progress.

CLIENT: “3 9” id=3. The client choose position 9.

SERVER: “3 0 1” id=3. The server pass since the game has ended. Status is client win.

CLIENT: “4 close” id=4. The client tells the server it has closed the connection.

Another Example:

CLIENT: “0 O” id=0. The client request to start a game with it playing ‘O’.

SERVER: “0 1 0” id=0. The server respond with position 1. Status is in progress.

CLIENT: “1 2” id=1. The client choose position 2.

SERVER: “1 4 0” id=1. The server choose position 4. Status is in progress.

CLIENT: “2 5” id=2. The client choose position 5.

SERVER: “2 7 2” id=2. The server choose position 7. The status is server win.

CLIENT: “3 close” id=3. The client tells the server it has closed the connection.