Tic Tac Toe game by s18749

The protocol

Commands:

Command	Sender	Description
PLAY	client	Informs the server that the client is willing to play.
MOVE x y	client	Reports the Client move (position)
LIST	client	Requests the list of active users (without itself). The list is returned immidiately after the LIST command and consists of strings in the format: uuid4 address:port, each followed by the newline. The list is followed by the END command that reports the end of the list.
LOGOUT	client	Informs the server that the user is no longer willing to play, which successes in the termination of the client subprocess of the server
ID uuid4	server	The response to the PLAY command. It accepts the client. Now the client is visible to others as waiting for the opponent
NO_OPPONENTS [10-0]	server	Notifies the Client that no other active users were found that could be matched as opponents. The command may be followed by the timeout in seconds till the server will stop making attempts to match the players. When sent without the timeout, informs about the end of the matching attempts.
ABORTED	server	Sent as a response to the logout command or when server aborts the connection for some reason (current implementation of the client does not care about aborts during the play, so the program may hang when used not properly)
END	server	Informs the client about the end of the list command.
START 1/0 CROSS 1/0	server	Informs the client about the match and the start of the duel. **START** keyword followed by **0** or **1** specifies if player should start the play. **CROSS** keyword followed by **0** or **1** specifies if the user is an **X** or **O**

Command	Sender	Description
UNRESOLVED last-move-x last-move-y	server	Informs the player about the end of the game due to last move (described by las-move-x and last-move-y) with the state unresolved
WIN win-sequence [last-move-x last-move-y]	server	Informs the player about the end of the duel that is resolved with the win status and the win-sequence in the format: number-number-number concatenated without any separator. each of the three numbers is the index of the winning field in the 9 elements table describing the state of the game
LOSE win-sequence [last-move-x last-move-y]	server	Informs the player about the end of the duel that is resolved with the lose status and the win-sequence in the format described in the **WIN** command
STATE x y now-moving	server	Informs the player about the opponent's move at the position x, y and specifies if the player can make a move or not
uuid4 uuid4 game-model win-sequence/false	server/broadcast	This command is used only to broadcast the state of the game to the viewer processes and is not a part of the interactive player comunication. The game-model is in the format of array of size 9 presenting the table red in rows from left top to right bottom.

Protocol flow

The server needs to be running before any comunication happens. The Player starts the comunication with the command ***PLAY***, that registers it on the server.

Server responds to the ***PLAY*** command with the ***ID*** command followed by the uuid4 id assigned for the player.

After receiving the id, Client starts listening for the Server ***START*** or ***NO_OPPONENTS*** commands. When it receives the ***NO_OPPONENTS*** command with the timeout, it displays the information that the server is looking for the match. When the same command arrives without the timer, it means that the match was not found, and player logouts and returns to the state before receiving the id.

If the match was found the server responds with the ***START-CROSS*** command and the duel begins.

The user that was selected (randomly with the probability of 50%) to start the game is unloacked and the player thread is suspended untill the interaction is registered.

Interaction wakes the player thread and starts the sync process.

The sync process consist of notifying the Server about the move (***MOVE*** command) and waiting for the respone.

The response may be of type ***WIN|LOSE|UNRESOLVED*** or ***STATE***.

If the state is one of the ending states, the winning sequence is drew on the screen and the screen is locked. The player can use the menu (...) to start the new game or exit.

If the response is of the type ***STATE***, the opponent move is drew to the screen and the screen is unlocked in respect to the command payload.

Broadcast

With each change of the game state, the server broadcasts it's new state to the audience using ports (given as the parameters) and broadcast adresses for each active (non-loopback) interface of the device. The broadcasted game state is sent over the UDP protocol.

The viewer processes may receive the broadcasted packets and view the duel's state in real time.

Viewer processes start listening to the packets sent from the port given as the parameter to the viewer process.

Duels are assigned to the broadcast ports in the order they request the broadcast method. If there are more duels than ports given to the server, only first n duels may be observed. (n -> number of ports)

```
Server port(TCP comunication) -b port(UDP) port(UDP)
            |                        |          |
            |                       /\         /\
            |                      /  \       /  \
        interactive               viewers    viewers
          player
```

How to compile and run

Linux

```
cd project/dir
chmod +x compile_linux run_linux

# server
./compile_linux server

# player
./compile_linux player
```

Run

```
# run server
./run_linux -t server -a "port -b broadcast_port broadcast_port ..."

# open x active players
./run_linux -n x -t player -a "server_ip server_port"

# open x viewers
./run_linux -n x -t player -a "-v broadcast_port"

# kill all instances and server
./kill_all
```