Network programming project (tetris) report

Pablo Rauzy
with Antoine Amarilli, Yoann Bourse and Marc Jeanmougin

January 11th, 2011

1 The protocol

We designed our protocol with simplicity (and fun) in mind, thus it is not able to avoid cheaters or anything, but is very simple to implement.

It's a plain text protocol using lolspeak grammar.

A server is running continuously. Clients connect to it when they want to play, and then they can start a new game. When the game starts the connection to the server is closed and a ring is formed between the clients. When a client lose, it notify its two neighbors to connect to each other so it can leave the ring while keeping it consistent. The last player staying in the ring is the winner.

1.1 The client / server part

First, the client need to introduce himself, before that, anything it says is ignored and the server always respond with:
server: `DUNNO WHO JOO IZ, WHO R U?`
to introduce itself, a client has to send:
client: `HOHAI!! I IZ IN UR GAME, LISTENIN ON` *`port`*`, MAH NAYMZ` *`name`*
to which the server respond:
server: `WELCUM` *`name`*`, PREPARE YRSELF TO ENTER TEH TETRIS!!`
and then the server sends to all connected client the new players list (see later).
In case the *name* provided by the client is already taken by someone on the server, the server responds with:
server: `ALL UR NAMZ R BELONG TO SUMONE ELSE!!!`
and the client isn't introduced.

Once connected and introduced the client must ping the server at least every two seconds. It does so by sending:
client: `I CAN HAZ A PONG?`

to which the server will answer:
server: LOLYES.

Before the game start, all connected (and introduced) client can chat by sending to the server:
client: I SAYZ *msg*
to which the server reacts by sending to everyone:
server: WUT PPL SAYZ *name*\t*msg*.

The first introduced client is considered the master player and can start the game whenever it wants by sending:
client: I CAN HAZ A TETRIS!!!
to which the server reacts by sending to everyone:
server: TEH TETRIS WILL START SOON
server: SEED R *seed*
and then the server sends to each client the ip adress and listening port of both of their neighbors:
server: U PLAYS WIF *ip1*:*port1* AT STARBOARD N *ip2*:*port2* AT LARBOARD
to which each client will respond:
client: KTHXBYE
to notify the server they have received the message. The server then close the connections.

Before starting the game, any client can quit by saying:
client: ME CHOOSE TEH RED PILL, KTHXBYE
to which the server reacts by sending the new players list to everyone (the same thing happen if a client stop pinging).

The players list is sent like this:
server: PLAYERZ LIST R:
and then for each connected (and introduced) client:
server: *name*\t*ip*:*port*
and then a blank line terminating the list:
server: .

If at any moment the server receives something else than what we just described from a client it responds:
server: DOES NOT COMPUTE. WUT DYA MEEN?.

This is all for the server.

1.2 The client / client part

The ring formed by the clients is oriented to the right ()STARBOARD: each client connects to its right neighbor.

The tetris board is 10 cells wide and 20 cells high.

In order to have everyone receiving blocks in the same order, everyclient must use the following pseudo random number generator, seeding it with the *seed* sent by the server: $r_{n+1} = r_n \times 15731 \mod 32003$.

The blocks start falling at the speed of one cell / second. Each ten lines this speed is multiplied by $1/0.9$.

When one drop a block, if it completes (and thus removes) some lines, the *combo* is incremented, if not, the *combo* is reseted to zero.

There is an attack/defense system which consist in sending penalty lines to one's right neighbor and trying to defend oneself from one's left neighbor. Each time one drop a block the following happen: if it completes (and thus removes) lines, then let p be the number of completed lines minus one added to the current *combo* value divided by two. Then a penalty of p lines is sent to one's right neighbor who add it to his penalty queue (attack). However if one have received penalities, then those penalties are deduced from p (defense) and the rest is either kept in the penalty queue (if p was too small) or sent to the right neighbor (if p was bigger).
When a dropped block doesn't complete lines, the penalty queue is emptied and for each penalty p, a random number n between 1 and 10 is choosen and p lines with the n^{th} cell empty are added at the bottom of the board.

Now, here is the protocol to do all this.

When connected to its right neighbor, a client introduces itself by saying:
client: `HOHAI!!!  I IZ` *name*.
When it receives this message form it's left neighbor, it answers:
client: `GRT! ME WUZ WAITIN 4 U TO PLAY WIF!!  MAH NAYMZ` *name*.

In order to be able to display one's neighbors board the clients send their board state to their two neighbors each time a block is dropped on it. The board is sent in an ASCII art representation:
client: `MAH BORD LOOKZ LIEK DIS`
and then 20 lines of 10 characters are sent, followed by a blank line. If a cell is empty it is represented by a dot (`.`), otherwise it is represented by a letter which depends on the type of tetris block that fills it: `i`, `s`, `z`, `j`, `l`, `t`, `o`, or a `p` if it's a penality brick.
When receiving a board, a client respond with:
client: `KTHXBYE`.

Penalties are sent like this:
client: `YO DAWG, I HEARD U LIEK PENALTIEZ! HERE IZ` *lines* `ONES 4 U!1!`.

If a client lose or want to leave the game it notifies both of its neightbors like this:
left neighbor:

client: I LOST TEH GAME, ALL YUR STARBOARD ARE BELONG TO *ip*:*port* NAO
right neighbor:
client: I LOST TEH GAME, ALL YUR LARBOARD ARE BELONG TO *ip*:*port* NAO.

2 Development

I choose to use Racket (`http://racket-lang.org/`) to develop the server and the client. Racket is a Lisp language descendant of Scheme. Thus it is very flexible and expressive, allowing fast and effective development.

I knew a bit of Scheme before starting the project, but this was my first real project in this language. Now that I've done it, I must say it was a really good choice.

2.1 The server

The server was written pretty quickly once I read the "network" part of the Racket guide.

The server spawn a new thread for each client.

The server's robustness could be improved a bit by catching all thrown exceptions (Gotta catch 'em all!) and manage them correctly. Currently if a client have a strange behavior (Marc's one did, I couldn't reproduce the situation) involving lots of EOF my server crash.

2.2 The client

The client is decomposed in four modules: one for the GUI, one handling client/server connection and protocol, one for handling the client/client protocol, and one for the tetris game itself.

The client have 4 threads, one for the GUI, one for handling the server, and one for each peer clients it connects too.

The first working with game version of the client could play only one time, because of some dirty initialisation. The current version is ready for nights of gaming, and auto-reconnect to the server at the end of a game to allow fast revenge.

In order to avoid lag at the beginning of the game, the local server (the one which each client run to accept its left neighbor's connection) is launched when the client connect to the (distant) server, so it is already running when another client attempt to connect.

As much as possible I redraw only the needed part of the board to make this operation as fast as possible. There's still improvments to make here, like using double buffering to avoid a little blinking caused by the erasing of the board before redrawing.