Network programming

# Multiplayer tetris based on memes

Yoann Bourse
(with Antoine Amarilli, Marc Jeanmougin and Pablo Rauzy)

2010-2011 : Semestre 1

**Résumé**

This paper presents the underlying mechanisms of the tetris multiplayer game designed for the network programming project. We will emphasize on the various development choices, especially the rules and implementation choices. Security was not the main concern in the development of the software : we wanted to provide a fun tetris, full of jokes and internet memes from the source code and the protocol to the user interface.

## Table des matières

# 1 Protocol

We designed a verbose humorous protocol for the client-server and client-client interactions. Our concern was not security, nor preventing cheating, but to provide a simple and fun protocol. Ours was therefore based on lolcat's KITTEH ENGLISH, enabling any overseer of the exchange to witness a lolcat conversation.
Before quitting or loosing, the clients are supposed to send warnings through the communication channels. But should this fail, most of the messages are followed by an acknowledgement of receipt, enabling to detect disconnections and failures, although using TCP communication it is not required (we could have only monitored the transmissions failures, that is to say the exceptions in the language). This enabled us only not to rely on technicalities linked to the programming language or network protocol. Note that blocks of several lines (players list, board state...) are often ended by an empty line so as to easy check when it is over.

## 1.1 Client-server interaction

We will describe in a nutshell the main characteristics of the client-server interaction. The client has the responsibility to ensure the server of its liveness by pinging it regularly (every 2 seconds). This way, the server manages a player lists which it broadcast to all players at every change (connection or disconnection). The server forbids the double use of a name, notifies invalid commands and handles a simple discussion protocol. Only the first person connecting to the game loby is allowed to start the game, so as to avoid chaotic behaviours. When done so, the server will broadcast a random seed so as everybody has the same pieces generation, and will form the peer-to-peer ring to play.

## 1.2 Peer to peer ring details

The clients are organized during the game as an oriented ring. They both have two neighbours (starboard and larboard) :
- The client initiates a connection towards its starboard neighbour
- It acts as a server and waits for a connection from its larboard neighbour.
Everytime someone leaves the ring (loss or disconnection), it is reformed by linking together its neighbours. The ring is formed thanks to the server which attributes the neighbours of every client based on their order of connection.

## 1.3 Client-client interaction

Once the ring is formed, or after any modification, clients proceed to a formal introduction of each other as a verification and an update (the client is not expected to keep track of the whole players list).

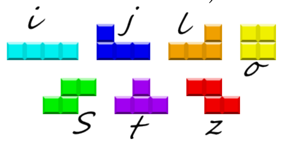
As we favored fun over performance, we chose a clearly suboptimal way to communicate the state of the game between players, that is to say broadcast an ASCII art version of the tetris board. This is clearly more information than required and does nothing to prevent cheating.
When leaving or loosing, the client is supposed to notify its neighbours of their new neighbours so they can reform the ring in a distributed fashion. Note that this does not enable the handling of sudden deconnection without warning or simultaneous deconnection of two neighbours. Those problems could not even be handled by keeping the initial players list because the state of the far away players is not updated in this protocol. Some changes would have to be made to handle these cases that we decided to neglect.

## 2 Rules

We will explain briefly the choices we made in the rules of this multiplayer tetris. We play on a 10x20 board.

### 2.1 Pieces handling

Pieces are generated thanks to a pseudo-random sequence ($u_{n+1} = 15731 * u_n mod 32003$) starting from the seed randomly generated by the server. The result is evaluated modulo 7 to give one of the 7 piece type (0 corresponds to i for instance) in the following order :



Once generated, the pieces fall at the rythm of one block per second. This rythm will be accelerated (cooldown reduced to 90%) every time 10 lines are broken, providing a slow rythm but a progression during the games.

### 2.2 Penalties

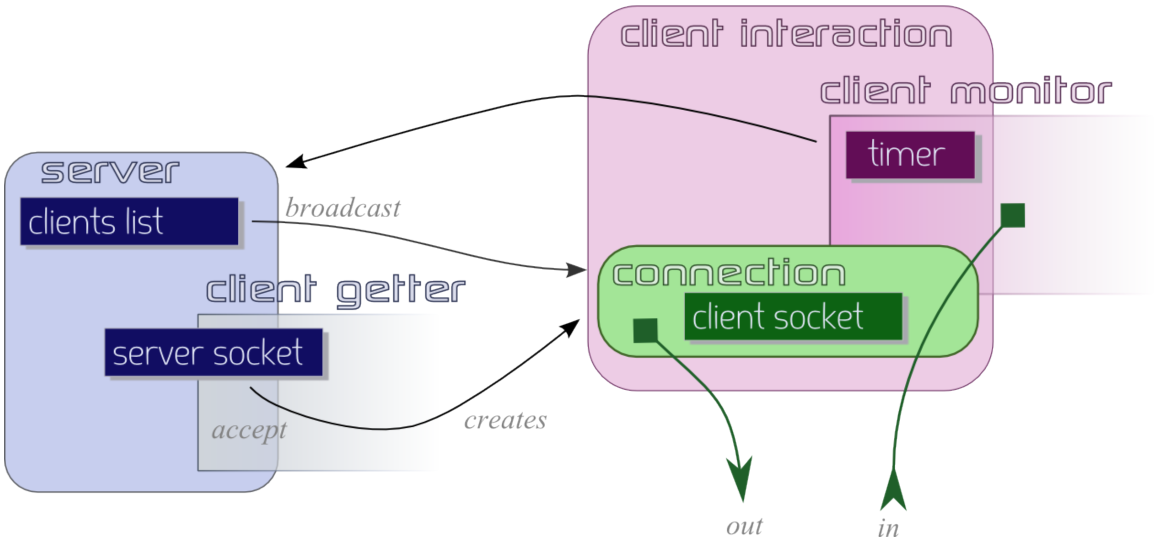
The multiplayer dimension of the games relies on penalties sent when lines are broken. Those penalties are sent only to the starboard neighbour. The penalty system uses combo points, which are equal to the number of pieces which have successively broken lines. That is to say this number is incremented every time one piece breaks line, and is reset every time a piece falls without breaking anything. Then, when $n$ lines are broken at once,
$(n-1) + \lfloor combo/2 \rfloor$ penalties are sent to starboard.
Therefore, several penalties can be sent at the same time. A penalty will result in an additional line popping at the bottom of the board, with only a single hole in it. When several penalties are sent at once, the hole in the block of lines that pop is synchronized to create a kind of well.

Moreover, received penalties are buffered and does not apply right away : if one line is done and penalties have been received, instead of sending directly $p$ penalties to starboard, we remove $p$ received incoming penalties from the buffer, and only then do we start to attack. If $p > b$, size of the buffer, we clear the buffer and send $p - b$ penalties. This system enables players to defend themselves, as the penalties are therefore only cashed when a piece falls without breaking a line.

## 3 Implementation

We will briefly discuss some implementation choices, using schematic drawings as support, in order to simplify the global structure of the program which is kind of complex (22 classes for the client and 6 for the server).

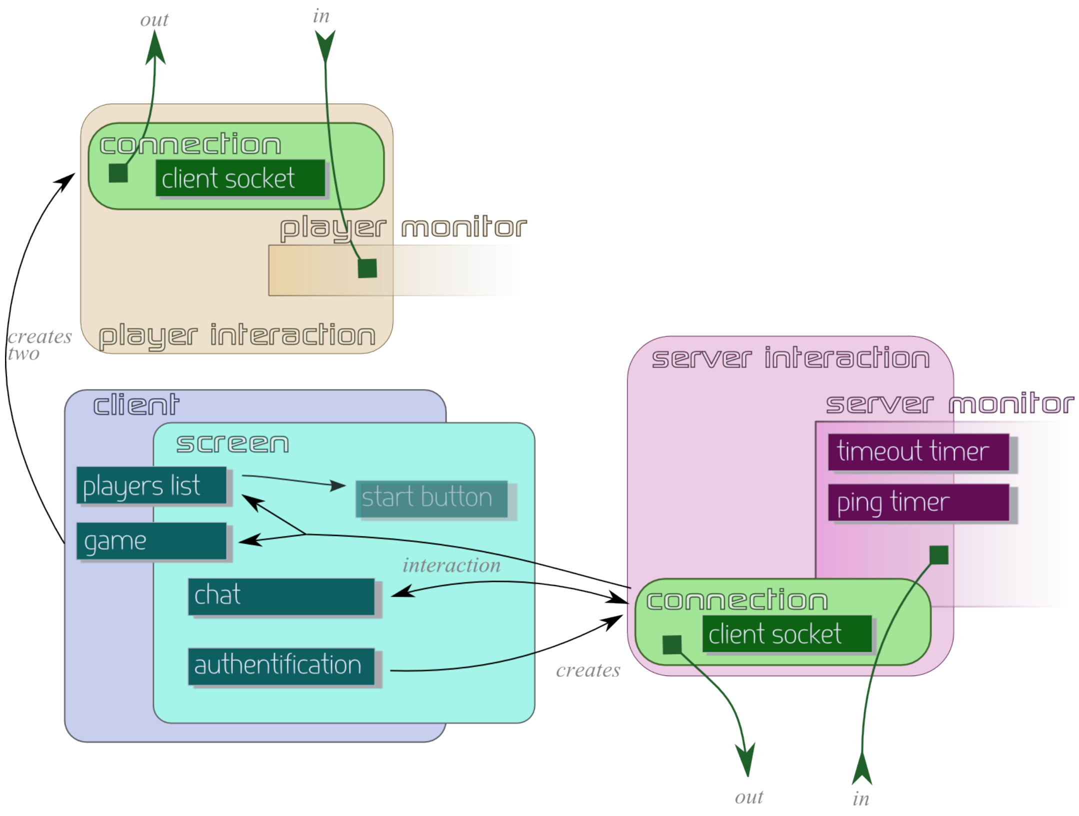
### 3.1 Server



The server relies on one client getter thread who accepts the connection requests and creates a thread per client, monitoring its liveness. This monitor also handles the reception of any incoming message. Those client monitoring thread are associated to the client interaction class which keeps all the data and handles the logic of the interaction. A players list is maintained all along.

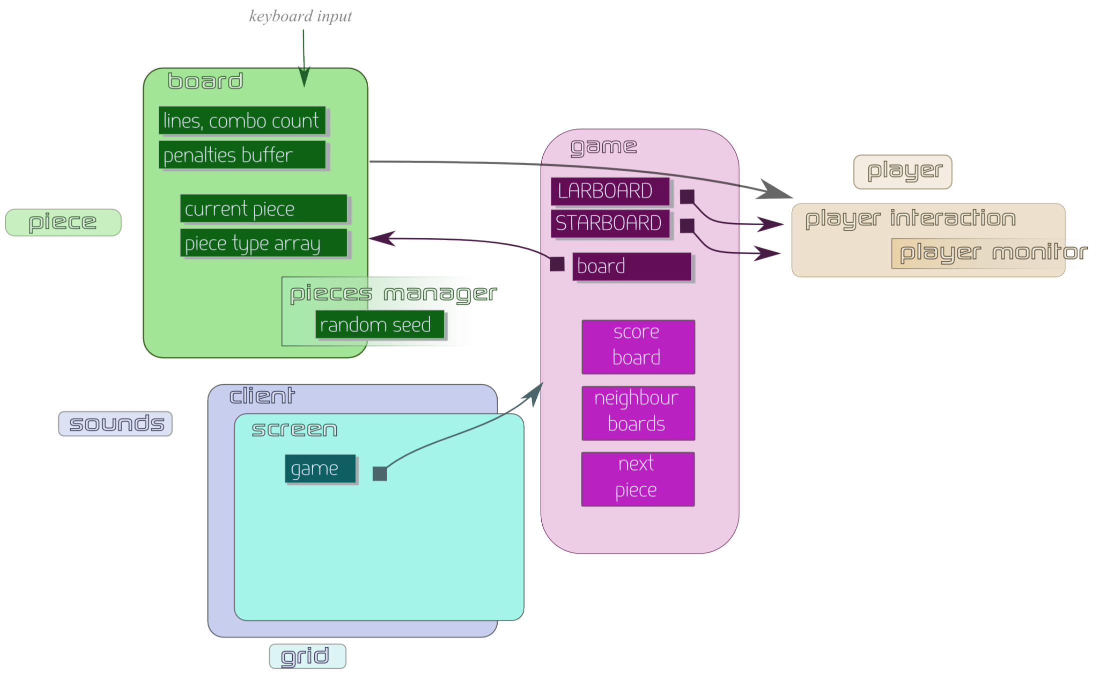
## 3.2 Client

### 3.2.1 Network interaction



The client go from a phase of communication with the server to a peer to peer communication. Each relies on the normalized class communication, providing higher level interactions than socket. The server monitors pings the server to inform it of its liveness, but also verifies that the server is still alive. In the second part, clients interact with each other through the same principle : the monitor thread handles incoming messages and the liveness checker via acknowledgement of receipts, and the interaction class contains the logic and the parameters of the connection. This class adapts itself to the role of the player : it becomes a server to wait for a connection from larboard or initializes a connection to starboard. Everytime a player leaves or looses, an new player interaction is created to replace it.
The core of the program is the screen class which handles all the graphical objects (chat, authentification module...) which pilot the whole program. This is also a window listener intercepting the closing of the game in order to shutdown properly (warning the server or neighbours).

### 3.2.2 Game



The game itself also depends on the screen class, which contains a game element. This graphical component centralizes all the data of the game : the graphical displays, the player interactions and the playing board. This latter is an event listener reacting to keyboard inputs. It is where the logic of the game actually takes place. A separate thread handles the pseudo-random generator, while a class piece enables the easy manipulation of the piece (which is, inside this class, in its own coordinate system.
The board contains a single moving piece (the current piece) and an array of piece types which describe the static part of the board. Interactions between the piece and this array are governed by several functions so as to fusion the piece with this array when it arrives at the bottom.
The penalty buffer is implemented as an integer queue where each elements represents a malus line (the integer is the position of the hole).
A sound class handles sounds effect through static functions. A grid class centralizes the drawing of all the grids (play board, next piece board or neighbour boards).

## 4 Work to be done

This version is mostly self-sufficient, but a few improvements can be done, especially in the protocol, in order to treat violent disconnections. Integrating the server to the client so that the first player hostes the tetris would make it a lot simpler to use.