Practical Work #4 – **Problem**: Tetris-like game

The aim of the problem is to achieve a **small network game like Tetris**.

To achieve this, a client-server model will be used where the network layer will be provided by **Sockets**. On the server side, multi-client management will be provided by **Threads**.

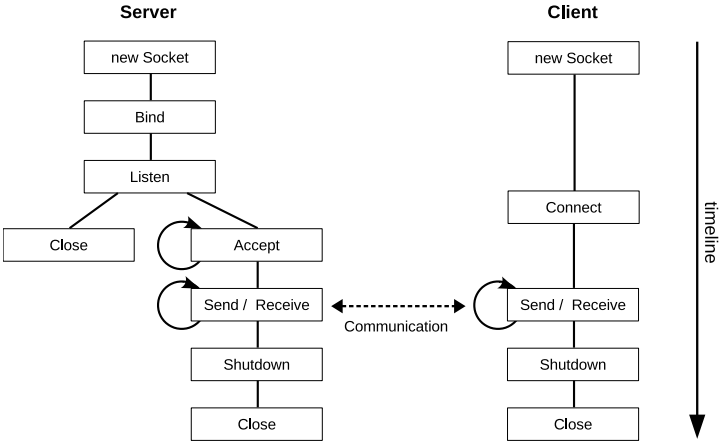
> This work is to be done in pairs and it will be graded. The assessment will mainly take into account the quality of your object-oriented code, a good use of Threads, and a relevant protocol for data exchange between the server and the clients.

Sockets

A network socket (or simply a "socket") endpoint of a bidirectional inter-process communication flow across an Internet protocol-based computer network. In practice "socket" usually refers to a socket in an Internet Protocol (IP) network, in particular for the TCP, which is a protocol for one-to-one connections.

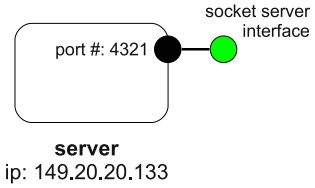
Socket states in the client-server model The server create sockets on start up that are in listening state. These sockets are waiting for initiatives from client programs. It may serve several clients concurrently, by creating a child process for each client and establishing a TCP connection between the child process and the client. Unique dedicated sockets are created for each synchronized connection.

The figure below summarizes the different states of a socket for a server and for a client.



Listening port *versus* Communication port

Step 1: Create and bind the socket on the server side. The server enters in the listening state.



Step 2: The client attemps to establish a TCP connection, which is accepted by the server.
Step 3: The connection is now possible on a specific communication port (transparently for the developer). So, the listening port is available again.

Samples in `C#`

Client side:
`https://docs.microsoft.com/en-us/dotnet/framework/network-programming/synchronous-client-socket-example`
Server side:
`https://docs.microsoft.com/en-us/dotnet/framework/network-programming/synchronous-server-socket-example`

Specifications

1. The server acts as a **producer** of blocks. Clients act as **consumers**.
2. The two available blocks are:

   `#` (takes the place of one square) and `##` `##` (takes the place of 2*2 squares)

3. When a player (client) creates N horizontal lines without gaps by placing a block,
   - for the player, the N lines get destroyed, and any block above the deleted line will fall.
   - for each other player, they receive N penalty filled horizontal lines at the bottom of their screens, and any block above go back. These penalty lines cannot be deleted; the blocks of such penalty lines will be displayed using star chars ('*').
4. At any time in the player's console are displayed
   - the number of lines he/she has already filled,
   - and the number of penalty lines for each other player.
5. The server will be started in the following console mode:
   `TetrisServer.exe listening_port number_of_columns maximum_number_of_lines delay_speed`
   e.g. `TetrisServer.exe 4321 7 20 1000`
   ✎ `delay_speed` represents the falling speed of the blocks (or the waiting times for the moving block to go down a notch)
6. A player (client) will be started in the following console mode:
   `TetrisPlayer.exe server_address server_port high_key right_key low_key left_key`
   e.g. `TetrisPlayer.exe 127.0.0.1 4321 z d s q`
7. When a player has a block positioned on its top line, he/she has lost. The socket communication is closed, the message "Game over" is displayed, and the client program stops.

✎ You can design the communication protocol of your choice.

HELP – On client side to manage keyboard event, you can just use an "active wait" like:

```
while(true)  // or other condition
{
   ConsoleKeyInfo key = Console.ReadKey();
   if (key.Key == ConsoleKey.Z ) // see ConsoleKey enumeration at msdn.microsoft.com
   {
      // ...
   }
}
```

Bonus

- Acceleration of time during game play.
- Use more kind of blocks, which can be rotated, like `###`.

Deliverables

The deliverables of the problem, to upload on Moodle before next practical work, are:

1. Your **code**.
2. A **short report** up to 4 pages indicating
   (a) client-server working
   (b) object design (*i.e.* class diagram(s))
   (c) protocol used
   $\Rightarrow$ to easily understand your code!