Session 2

Exercises

Implement a networked application with client/server topology using the TCP protocol.

NOTE: A skeleton of the application is provided in the virtual campus. As usual, the code will be split in two parts, the server and the client. The incomplete pieces of code are easily found by searching the string *TODO*. Mainly, you will have to implement the functions *handleIncomingData()* and *handleOutgoingData()* both in the server and in the client parts.

The goal of this session is learning how to use non-blocking mechanisms when working with sockets, as many games need a high degree of interactivity (performing at least at 60 frames per second, for instance), and thus, they cannot afford waiting for sockets until they are available to read/write data from/to the network.

In this network topology, the **server** is typically waiting for new connections through al listen socket, and receiving and sending game data from/to clients through connected sockets. It is quite clear that working with TCP sockets, the server will need to maintain several simultaneous connections with clients (that is, several connected sockets). **In order to perform non-blocking queries, the server will use the function select().**

The server application loop will perform the following operations:

- Handling incoming data
    - New connections through the listen socket (accept)
    - Receiving data from connected sockets (recv)
- Update the game state (empty functions)
    - AI, physics simulation, ...
- Handle outgoing data
    - Send data through connected sockets (send outputBuffer)

Regarding the **client**, the implementation will not be too different with respect to the previous' class exercise. The client will only work with a unique socket connected to the server. Therefore, instead of using the function *select* (like the server code does), it will be easier **to configure this socket in non-blocking mode.**

The client application loop will perform the following operations:

- Handling incoming data
    - Receiving data from the server (recv)
- Update the game state (empty functions)
    - Mainly rendering
- Handle outgoing data
    - Send data to the server (send outputBuffer)