Ejercicio 6

El objetivo de este ejercicio es programar las bases de un chat cliente/servidor sobre el protocolo UDP.

Organización de la solución

Debes crear una solución con dos proyectos:

- Uno para el servidor
- Otro para el cliente

Si vamos a utilizar cliente/servidor, el código de servidor va a ser más complejo que en los ejercicios de la parte de P2P. Mejor tenerlo en un proyecto separado.

Código de servidor

1) Debe preparar un socket UDP que se vincule a un puerto y espere mensajes nuevos.
2) Cuando llegue un mensaje debe comprobar si es de un cliente nuevo o de un cliente ya existente.
    a. Si es de un cliente nuevo, anotará la ip y puerto desde donde le envía los mensajes
    b. Si es un cliente ya existente, no lo anotará
3) Se recoge el mensaje que ha llegado de este cliente y se reenvía a todos los demás clientes conectados (en este punto, puedes decidir si envías el mensaje también al mismo cliente o no)
4) Ten en cuenta que no estamos controlando desconexiones. En este ejercicio todavía no se os está pidiendo.

Código de cliente

1) Debe preparar un socket UDP para poder enviar y recibir mensajes del servidor
2) El servidor no va a saber que existimos hasta que le enviemos el primer mensaje. A partir de ese momento, pueden llegarnos mensajes del resto de clientes conectados al servidor.
3) El cliente debe estar preparado para poder enviar y recibir mensajes del servidor en cualquier momento.

Desvincula la librería de red

Igual que hicimos en TCP, pero ya desde el principio, vamos a desvincular nuestro código de la librería SFML. En TCP se os pasó una posible estructuración para TcpSocket, TcpListener y SocketSelector. En UDP necesitamos solamente el UdpSocket.

La implementación que se propuso para TcpSocket fue la siguiente:

```
class TcpSocket
{
    sf::TcpSocket* tcpSocket;
public:
    TcpSocket();
    TcpSocket(sf::TcpSocket* _tcpSocket);
    ~TcpSocket();

    sf::TcpSocket* GetSocket();
    void SetSocket(sf::TcpSocket* _tcpSocket);
    std::string GetRemoteIP();
    Port GetRemotePort();
    Port GetLocalPort();
    Status Connect(std::string _ip, Port _port);
    InputMemoryStream*  Receive();
    Status Send(OutputMemoryStream& _oms);

    void Disconnect();
};
```

Se pide adaptar esta clase para UdpSocket de forma que la lógica del chat esté desvinculada de la librería SFML.

Incorporar interfaz gráfica

La práctica que implementaréis en este segundo bloque sí que va a tener una mínima interfaz gráfica. Si tenemos que simular tiempo real, movimiento, etc. necesitamos interfaz gráfica.

Si bien podéis usar SDL si os sentís más cómodos, se os propone utilizar también SFML.

Tenéis disponible en Classlife el código base de un chat SFML que podéis retocar a vuestro gusto para este ejercicio. En hecho de utilizar una interfaz gráfica hace que no tengáis que tratar los cins/getline como un bloqueo más en un thread a parte. Es recomendable que incorporéis esta interfaz para ver cómo cambia el código.

Por otra parte, de la misma forma que hemos desvinculado la librería SFML de la parte de redes, se podría hacer lo mismo con la interfaz gráfica. Desde este ejercicio y desde la práctica final no se os va a pedir esta partir, pero a nivel de organización de código sería lo más recomendable.