# Lab session 4

Exercise

## Adding MemoryStream

Copy the following files available in the virtual campus into your project directory:

- *ByteSwap.h*
- *MemoryStream.h / MemoryStream.cpp*
- *Messages.h*

Add them to the project and:

- Include *Messages.h* at the top of all includes in *Networks.h*
- Include *MemoryStream.h* at the top of all includes in *Networks.h*
- Include *MemoryStream.cpp* at the top of all includes in *UnityBuild.cpp*
- Right click on *MemoryStream.cpp* and open the *Properties* panel, then set *Exclude from build (Yes)*.

## Introduce MemoryStreams in ModuleNetworking

Add the following static method in *ModuleNetworking*:

```
bool ModuleNetworking::sendPacket(const OutputMemoryStream & packet, SOCKET socket)
{
    int result = send(socket, packet.GetBufferPtr(), packet.GetSize(), 0);
    if (result == SOCKET_ERROR)
    {
        reportError("send");
        return false;
    }
    return true;
}
```

Modify the signature of *onSocketReceivedData* in *ModuleNetworking* and update the subclasses' overrides (in *ModuleNetworkingClient* and *ModuleNetworkingServer*):

```
virtual void onSocketReceivedData(SOCKET s, const InputMemoryStream &packet) = 0;
```

To adapt our code to this last change, go to *ModuleNetworking::preUpdate()*, and change the container used to receive the packet data:

```
InputMemoryStream packet;
int bytesRead = recv(socket, packet.GetBufferPtr(), packet.GetCapacity(), 0);

if (bytesRead > 0)
{
    packet.SetSize((uint32)bytesRead);
    onSocketReceivedData(socket, packet);
}
```

## Changes in the Hello message

Change the way we send the “Hello” message to start using *MemoryStream* to create packets:

- Add the message “Hello” in the *ClientMessage* enum in the file *Messages.h*.
- Modify the *ModuleNetworkingClient::update()* function to contain this code:

```
if (state == ClientState::Start)
{
    OutputMemoryStream packet;
    packet << ClientMessage::Hello;
    packet << playerName;

    if (sendPacket(packet, socket))
    {
        state = ClientState::Logging;
    }
    else
    {
        disconnect();
        state = ClientState::Stopped;
    }
}
```

Now we need to modify the code that receives those packets in the server side:

- Modify the *ModuleNetworkingServer::onSocketReceivedData()* this way:

```
ClientMessage clientMessage;
packet >> clientMessage;

if (clientMessage == ClientMessage::Hello)
{
    std::string playerName;
    packet >> playerName;

    for (auto &connectedSocket : connectedSockets)
    {
        if (connectedSocket.socket == socket)
        {
            connectedSocket.playerName = playerName;
        }
    }
}
```

## Welcome the client

After saving the player name, the server could send a welcome packet back to the client to let it know it has logged correctly. A welcome packet could contain some fields such as:

- Message type (mandatory)
- Welcome text
- Player color
- Etc..

## TODOs

Using the method explained to send packets through a socket, program a chat application that allows connected users to have a conversation. Among others, you can implement the following features:

- Send the welcome packet to the newly connected user.
- Send a non-welcome packet if the player name already exists.
- Use ImGui to type text and send messages to other users.
    - The client application will need to maintain a list of all received messages in order to print them with ImGui. Upon writing a new message, it is up to you whether to update the list right after pressing the return key or waiting until receiving the feedback from the server.
- Notify all users about the new client connections (e.g. *"Peter joined"*).
- Notify all users about any other client disconnection (e.g. *"Peter left"*).
- Special commands:
    - /help: to list all available commands.
    - /list: to list all users in the chat room.
    - /kick: to ban some other user from the chat.
    - /whisper: to send a message only to one user.
    - /change_name: to change your username.
    - etc
- Whatever comes to mind...

**NOTE:** I am using the words **user**, **player**, and **client** indifferently here.