A player connects to the server and joins the game: explanation

The application (ClientApp) boots up. The user gets asked the server IP through the user interface. Then, by clicking confirm, it gets sent to ClientApp. This will create a NetworkAdapter, which will try to connect to the server. If it fails, the failure will be reported to the user, and the IP will be asked again. NetworkAdapter will also create the objects responsiblefor reading from and writing to the socket. The first will also create an EventRegister, which is responsible to register for every event a function that gets called when it arrives. Then, NetworkAdapter will register for every event a function to handle it (for visual clarity, we only show one in the diagram).

Once the connection is established, we can either follow what happens on the client side, or what happens on the server side. Starting from the second: for every client that connects, the server creates two threads, one blocked on the read of the socket, the second blocked on the read of a queue (RequestQueue).

On the client side instead, a thread is created too, blocked on the read of the socket.

At this point, the connection is established, and the classes responsible for communication have been instantiated both on the server side and on the client side. So the client can send to the server the username of the player, after asking it to them, encoded in an event-object (NewPlayerEvent), after transforming it in a JSON. In this diagram, the network communication is represented by a brown arrow between the client and the server sockets.

The message gets read by the thread of the server that was blocked on the socked read, it gets transformed back into a POJO and inserted into the RequestQueue. Then the thread blocks again on the socket read.

At this point, the thread blocked on the read of RequestQueue extracts the event. Notice how ClientHandlerReciver, RequestQueue and RequestElaborator implements a typical case of Producer-Consumer.

The event that has just been extracted gets sent to the EventRegistry of the server. At this point, the handler for the event gets called. In this particular case, the handler will create a lobby with the player as the leader, and it will send to the client an event containing the state of the lobby. With this, the UI can show to the player the state of the lobby they're in.

While the creation of messages between client and server is the same, the reception of them is slightly different. The message is still read by the thread blocked in the socket read, but it isn't inserted in a queue, instead it gets immediatly handled. The threads blocks again on the raed.