Implemented Functionality

Function	**C#**	**Java**
Client establishes a connection with the server	Yes	Yes
Client is assigned a unique ID when joining the game	Yes	Yes
Client displays up-to-date information about the game state	Yes	Yes
Client allows passing the ball to another player	Yes	Yes
Server manages multiple client connections	Yes	Yes
Server accepts connections during the game	Yes	Yes
Server correctly handles clients leaving the game	Yes	Yes
Client is compatible with the server in the other language	Yes	Yes
Additional tasks:		
Client GUI	No	No
Server GUI	No	No
Server restarts are correctly implemented	No	No
Unit tests	No	No

Protocol

- It is a client-server application, i.e. the client initiates a connection;
- The protocol is text-based, i.e. not binary;
- When the connection is established, the server immediately sends information about the current state of the game to all connected clients;
- The protocol does not include any authentication, considering it is outside of the scope of the assignment;
- The only information the client sends to the server is the id of the client which it wants to pass the ball to. If the client doesn't have the ball, it just receives updates from the server;
- The only information the server sends is the current game state, which includes the client's id, the number of players connected, the list of players connected, and the id of the client that currently holds the ball. This information is sent to all connected clients everytime there is a change to this state, i.e. a client joins the server or the holder passes the ball;
- If there is an error the server closes the connection.

Client Request	**Server Response (to all clients)**
(On connection)	<Id> <Holder's id> <Number of players> <player1> <player2> ...
<client id>	<Id> <Holder's id> <Number of players> <player1> <player2> ...

Client Threads

Each Client process has 2 threads:

- **Main Thread:** Created when the process is started, this thread is responsible for the connecting, receiving and interpreting the information sent by the server, as well as managing the UI, which is console based. It terminates when the process terminates or when connection is lost.
- **ClientWriter Thread:** Created when a player has the ball, it awaits for the user to type the valid id of the player which will receive the ball, as well as sending this id to the server. This threads was developed so the main thread is not stuck waiting for input and can still update the UI information. It is terminated after the id is sent to the server.

Server Threads

The Server process has 2 threads:

- **Main Thread:** Created when the process is started, this thread is responsible for listening for connections in the designated port, and distribute sockets from the ServerSocket to the clients. It also makes sure the updates are sent by all handlers. It is terminated when the process terminates.
- **Handler Thread:** Created when a client successfully connects to the server, this thread is responsible for the communication with the client. It is terminated when the connection with the client is lost.

Project Review

The project, overall, wasn't by itself too difficult, but certainly time consuming. The most challenging part of it was understanding how the socket communication and data streams behaved, and what they could and could not do, considering I have never worked with that before. The translation of the program from Java to C# was surprisingly easier than estimated, as I have close to no experience with the latter, apart from the lab classes of this module. But the task proved to be not as hard as I anticipated, mainly due to the similarity between the syntaxes of the two languages.