Design choices

The major choice was whether to use sockets or java RMI. Since java RMI is much more powerful (for example the server can call callback methods on its clients), we quickly decided to go with RMI instead of low-level sockets.

How does the code work

- The MatchInterface is worth mentioning: we want to have matches as remote objects so that the two opponents can interact with their match directy. Like this, we do not clutter the server class with methods that are only related to matches.
- To do so, the MatchInterface extends *Remote* and the implementations extend *UnicastRemoteObject*
- There are two implementations: one for matches between two players and one for matches against the server. The beauty is that the client can treat a match on the interface level since the basic match logic is independent of the concrete match type.
- As an extra feature for matches between players, we added a thread to the client class that runs in the background and informs the attacking client in regular time intervals how much time is left to make an attack.

Work partition

The code was written in some sessions of pair programming, so all work was done together.