In this project we have developed:

- RMI
- Socket
- GUI
- CLI
- 2 advanced functionalities: restore session of the player, lobby and part of the User Management (after logging the first time, the players are saved on the server with their password, and they can enter again: in this way, nobody else can use their name even if they are offline)

At the beginning we have divided the game structure in two parts: client and server.

The server is divided essentially in these components:

- Model that saves both connected players with their status ("online", "in game", etc.), and the list of the games with their status
- Game's logic (GameManager) that modifies the model of the game according to the players' choices
- Controller (RMIControllerServer) that satisfies the client's requests (using the game's logic for the requests related to the actions of the game)
- Connection to the client (Connection, implemented by 2 class) in order to permit to the server to contact the client (through methods in RMI, while sending messages with a protocol in Socket)
- Receiving connection (SocketServer and RMIControllerServer, the latter plays two roles) that permits to the server to be in contact with the client (through methods in RMI, while sending messages with a protocol in Socket)
- Prepare data to be sent to the client (PrepareDataForClient): it is a class that reads the game's status from the model and passes it to the client, by using different elementary types of Java (principally String, int, etc..). In this way, we pass to the client only the data that are needed, namely the ones that are changed, and not the whole game. Moreover, the client will not need to be aware of all the complex model of the server, but it only needs to know how the data are organized in the constants or vectors that are passed by the server

Instead, the client is divided into:

- Model that saves:
    - Locally, the data of the match needed to the single players in order to play
    - In which screen the player is
    - Some information about the player (name and type of chosen connection)
- View, divided into CLI and GUI, that interacts with the user (visualizing information and reading insert commands)
- Controller that, according to the screen in which the player is and to the choice made by him, will execute some functions
- Connection to Server (ClientConnection implemented by 2 class) that permits to the client to contact the server (through the invocation of RMI methods, while sending messages with a protocol in Socket)
- Receiving connection (SocketClient and RMIClient) that permits to the client to be in contact with server (through methods in RMI, while sending messages with a protocol in Socket)

In order to make the View independent from the Controller, we have used the Observe-Observable pattern. Instead, in order to make the Controller independent from the Connection (both in the client and in the server) we have created an interface (Connection and ClientConnection) implemented in two ways by RMI and Socket.