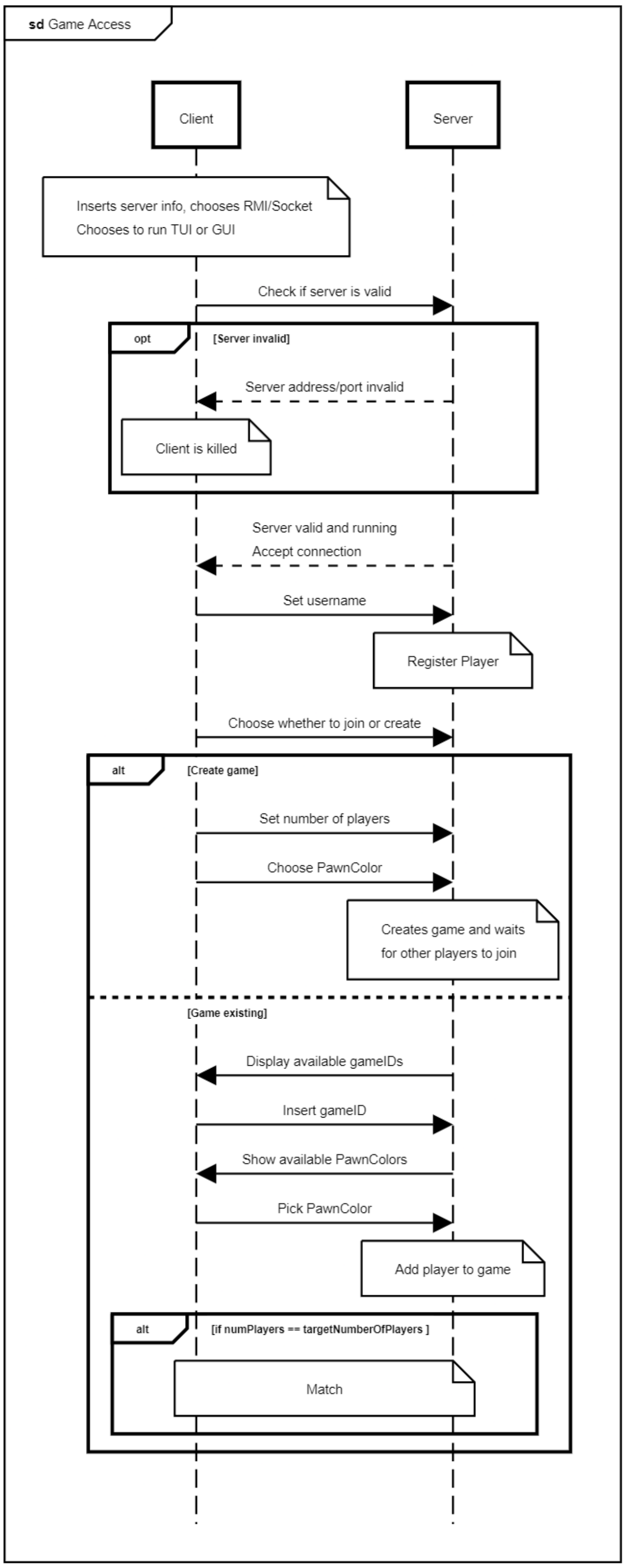
COMMUNICATION DESIGN – SEQUENCE DIAGRAMS

GAME ACCESS



The Game Access diagram shows all the phases, actions and network exchanges between the client and the server when a client starts the game executable. Firstly, the user inserts the server's IP address and port to connect to it, chooses whether to connect through RMI or Socket and, lastly, picks the preferred graphical interface between CLI and GUI.

The client then proceeds to verify if the server address and port are valid and open for a connection with the specified protocol chosen by the user; if not, the client is killed and the user must re-boot the executable and insert the correct information. Otherwise, the server accepts the connection from the client and the user is prompted to insert a username to be recognisable inside the game. This information is shared to the server.

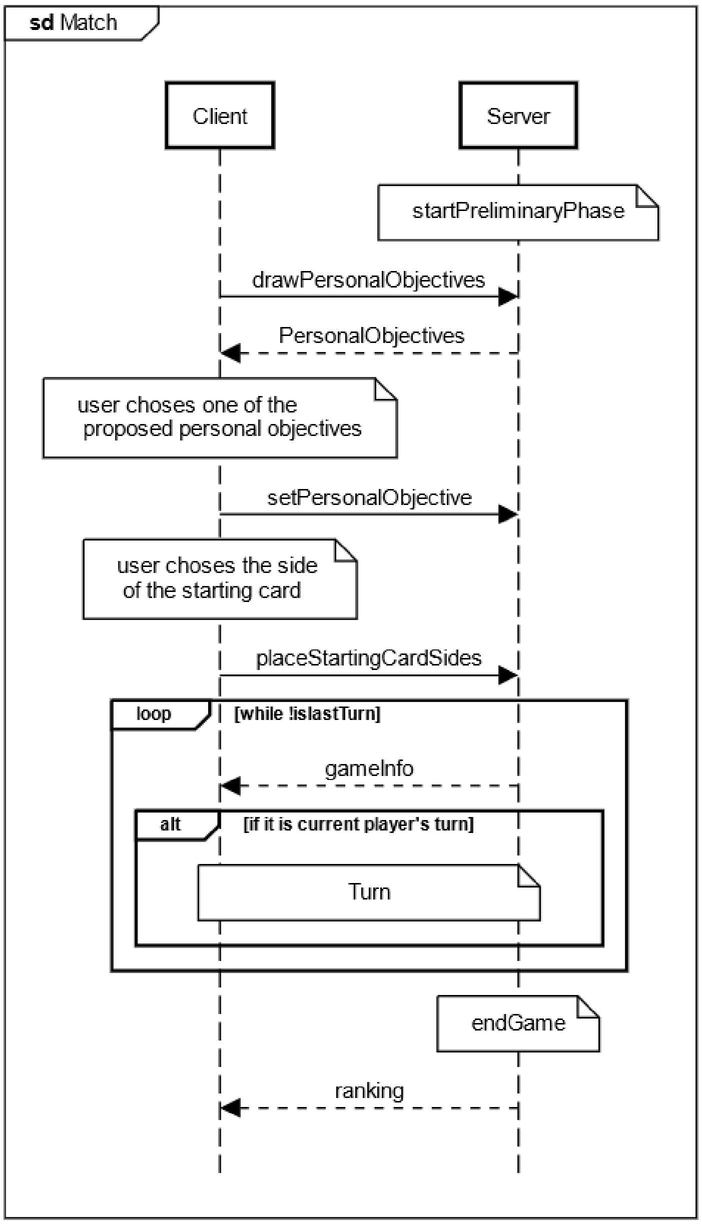
After the registration, the user has either the option to create a new game or to join an existing one on the server he's connected to.

- If the user wants to create a game, he's prompted to insert the number of players that must join the game and to pick his/her pawn colour; this information is forwarded to the server, which creates the game and waits for the remaining players to join.

- Instead, if there are already some games running, the user might want to join one, so the server provides the client with all the available game IDs of ongoing games. As soon as the user inserts a valid ID to join that game, the client is given the remaining pawn colours to choose from; after the user has chosen the colour, the information is forwarded to the server which adds the player to the game.

Every time a player is added to the game, the server checks if the selected number of players for that match is reached: in that case the game is started.

MATCH



The diagram describes the interaction between the client and the server during the match phases.

At the start of the match, the server executes the preliminary phase which includes:

- dealing the starting cards to the players
- dealing the hands cards to the players
- draw the common objectives of the match

Then, for each player (in the randomly chosen order), the client draws two possible personal objectives (*drawPersonalObjectives*()) and choses one of them. Then it sets the chosen personal objective calling the specific server method (*setPersonalObjective*()).

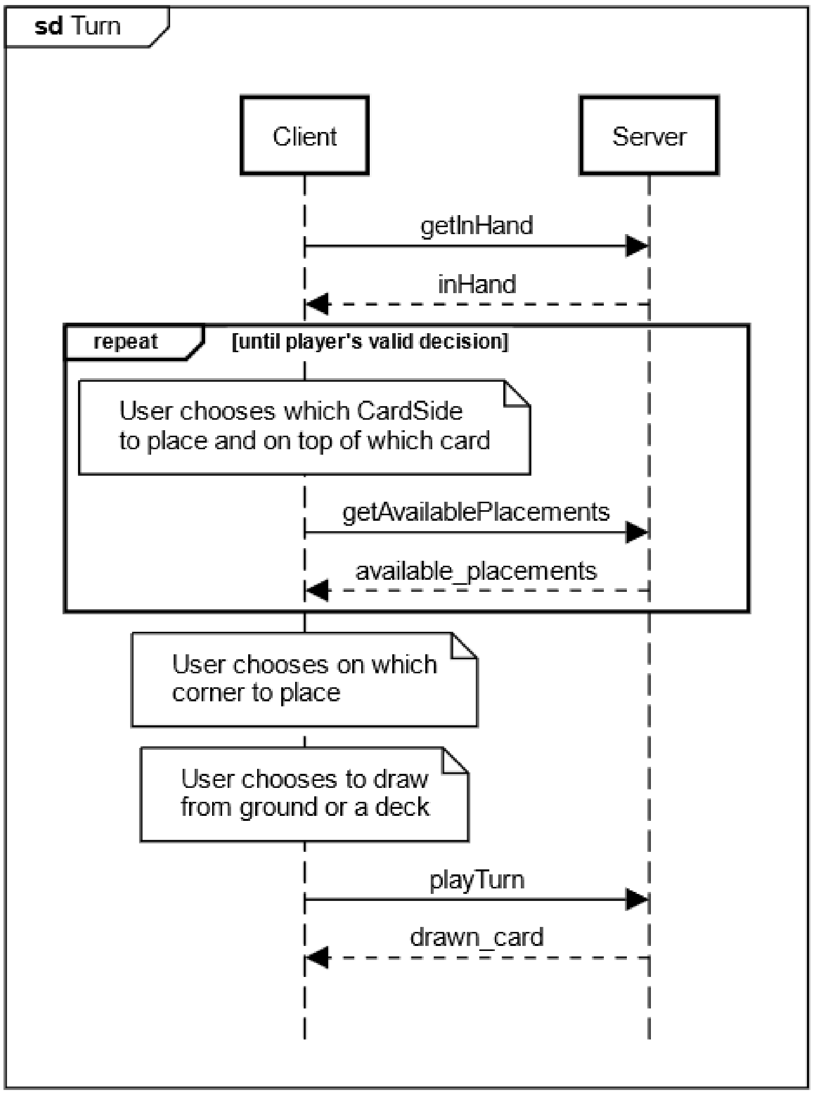
Finally, each player places the starting card he has drawn so to show the desired side (*placeStartingCardSide*()). This concludes the initial steps of the match.

Now, while no player has triggered a *lastTurn* event (no player has reached 20 points and there are still cards to be drawn), each player receives from the Server information about the game (*gameInfo*). It entails: scoreboard (the updated scores of all players), situation of Decks (cards laying visibly on ground and the possibility draw) and the placed cards of the opponent who has just played (such that clients can update their locally saved situation).

The current player plays its own turn according to the Turn sequence diagram.

Once the *lastTurn* event is triggered and every player has played the same number of turns, the *endGame* method is called. This method computes the points of each player according to the points he collected during the match from the deployed cards and the ones given by the personal and common objectives. Finally, this method computes a ranking of all the players in the match (breaking parities according to the game rules) and returns the ranking which is then displayed.

TURN


sd Turn
Client
Server
getInHand
inHand
repeat
[until player's valid decision]
User chooses which CardSide to place and on top of which card
getAvailablePlacements
available_placements
User chooses on which corner to place
User chooses to draw from ground or a deck
playTurn
drawn_card

This diagram describes the interaction between Client and Server during the enrollment of a player's turn.

At the beginning of the turn, the player that is playing asks the server for his/her in-hand situation (*getInHand()*) and after receiving it the View displays all the gathered information.

Now, the Client is asked to choose which CardSide he/she wants to place and on which of the deployed cards.

The Server is asked to analyze the situation of the player (*getAvailablePlacements()*), checking the corners of the card on which he/she wants to place and telling if they are available for the placement. The corresponding indexes are then returned to the client.

In this way:

- the player knows which placement are legal for the selected couple of cards (CardSide to place - card on which to place)
- we guarantee that the Client won't make any illegal placement and so we are avoiding extra requests to the Server.

This process of selection and *getAvailablePlacements()* can be repeated by the player until he/she reaches a decision of what move to perform.

Finally, the user is asked to choose how he/she wants to draw a new card (type of card and from where). Now that both the placement and the draw have been decided, a request is sent to the Server containing these information through the method *playTurn()*. The server finalizes the move performed by the player, updating his/her deployment, hand, resources and score. As a response, it gives back the drawn card to the Client, so that it can update the locally saved situation.