Communication protocol

AM36 – Caliandro Converso D'Antini

Our communication protocol is implemented by following classes:

- ServerMain
- ClientHandler
- ClientMain

When the connection is initialized, the server delegates the process to a thread, previously initialized by a thread pool, that runs an instance of the ClientHandler class. Each ClientHandler object is associated to a VirtualView, which is a game room used to group the connected clients into different game. It contains a List of clients that are not logged in (waitingList), and a Map with those that are (clientMap). The first client to enter in the waitingList is marked as the firstPlayer.

All the message exchanged between client and server are java objects that implement an interface called Message that extends Serializable, eventually containing arguments as attributes. The *visitor* pattern is used to read these messages: for each Message subclass there is a method in a specific MessageVisitor class.

Before any message exchange the sender (Client or Server) sends a ping object and waits a pong response (as a ping object) to ensure that the connection is on.

When the user makes requests, it must type a command following this format:

```
identifier: arg1, arg2, …
```

afterwards the command is converted in an object (implementing Message class) based on the identifier that is built with the appropriate arguments.

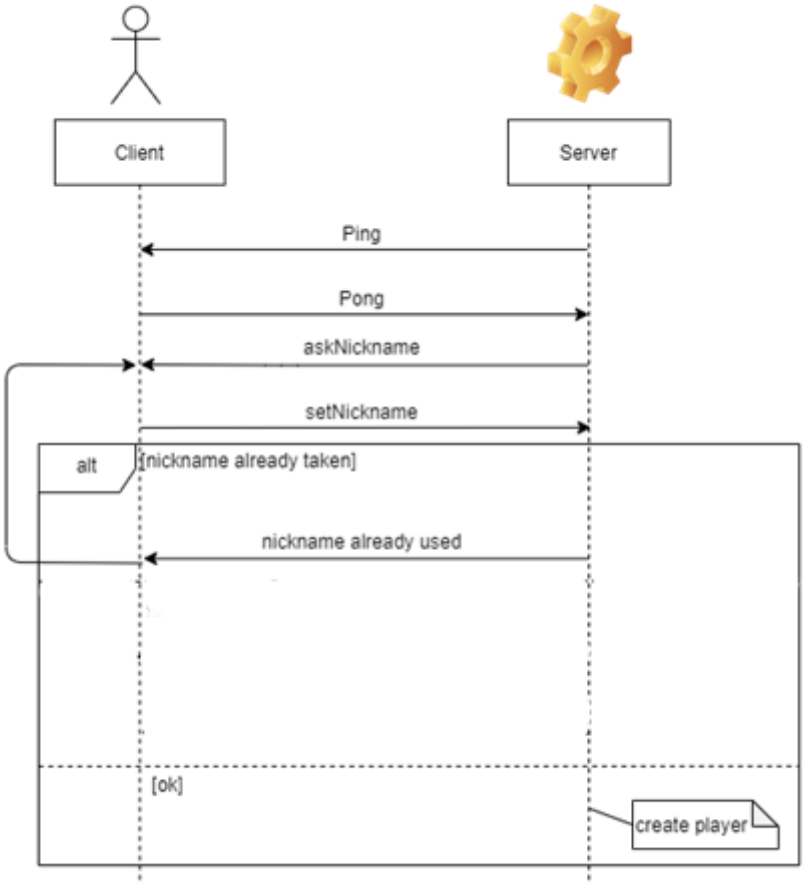
The syntactic correctness of the command is controlled by the Client itself. It also verifies the matching with an expected command.

Instead, the semantic correctness of a message is controlled by the Model part of the server.

The whole communication can be divided into the following phases:

- Login phase
- Game creation/join phase
- Action phases
- End game phase

- Login Phase:



(Here and only here the ping-pong exchange is explicit)

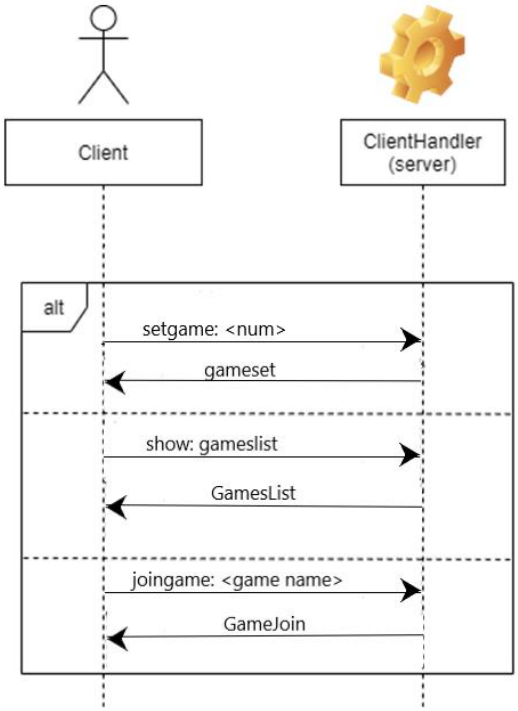
The alt frame shows the different responses of the server in each case, like in sequence diagrams.

In the Login Phase, after a ping exchange, the server sends to the client a request for a nickname.

The client has to send a SetNickname message (containing the nickname as attribute) by typing the command *SETNICK: <nickname>*.

If the nickname has been already taken, the user has to redo the command. If the nickname is available, it's checked if the player was in an active game, and in that case is re-added to that game: otherwise, the player enters into the Game creation/joining phase.

- Game creation/joining Phase:

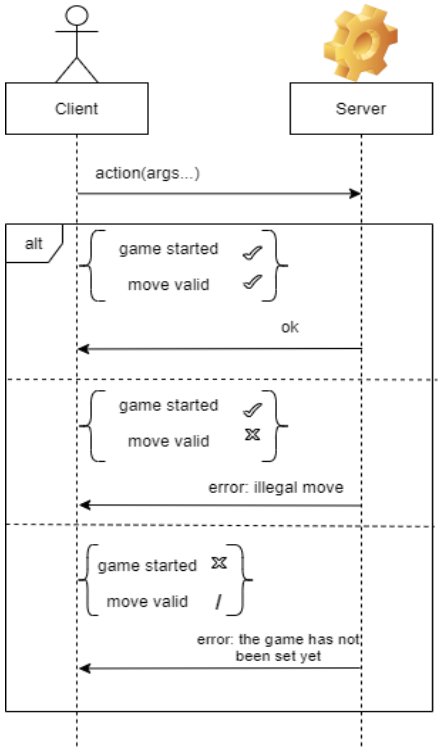


In the Game Creation/joining phase, the client can choose to do one of two possible moves:

- Create a game: *SETGAME: <num>*.
- Join a game: *JOINGAME: <name>*.

To obtain a list of available games, client can use the command *SHOW: gameslist*, that returns, for each available game, the name, the number of connected players and the max number of allowed players.

- Action Phase:



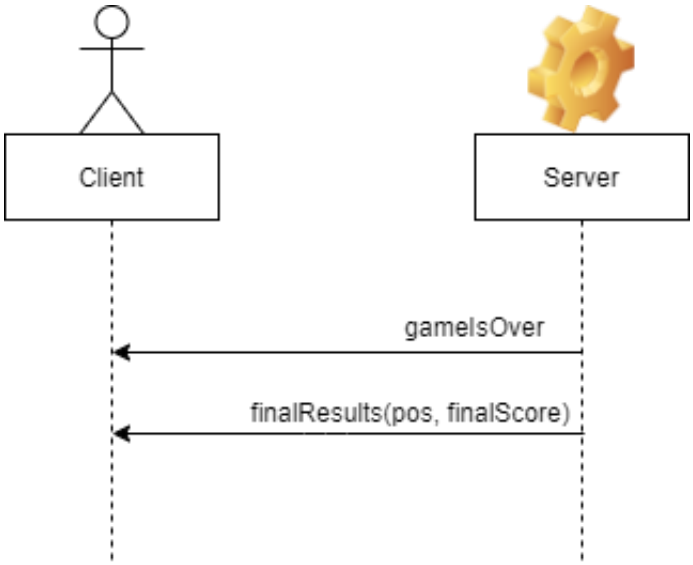
This phase is similar for all the moves that the user can do during the game.

The list of moves is the following:

Command syntax	Description
DEPLOYRES: <resource, depot>	Inserts the resource in the hand of the player into the specified depot.
TAKERES: <depot>	Takes a resource from the specified depot and puts it in the player's hand.
GETRES: <resource>	Gets a resource in the resource hand.
DISCARDLEADER: <pos>	Removes the leader card from the "POS" position in the hand of the player.
USELEADER: <pos>	Deploys (and activates) the leader card from the "POS position in the hand of the player.
CHOOSE: <turnphase>	Chooses the next turn phase to play.
BUYDEVCARD: <level, color, space, stores[]>	Buys a card from the development cards grid using resources from the indicated stores.
USEMARKET: <r/c, pos>	Chooses the row (or the column) of the market tray to get resources from.
DISCARDRES: <resource>	Discards the resource indicated from the hand of the player.
ACTIVATEPROD: <pos, cost[]> (default) *ACTIVATEPROD: <pos, store, product>* (leaders) *ACTIVATEPROD: <pos, cost[], product, stores[]>* (board)	Activates the production indicated by pos, using the stores indicated in cost[] (or in STORES[] when the production is the board one).
NEXT	Passes to the next turn phase.
BACK	Goes back to the previous turn phase.

The client sends a specific message for each move, the server receives it and, if the move is valid, the server sends an Ok message to the client; otherwise, refuses the message and sends an error message.

- End Game Phase



When the game is “over” (a player reaches an end game condition), the server sends a GameIsOver message to all the clients of the game.

When, instead, the game is “finished” (the last player has done his turn), the server sends a FinalResults message to all the clients of the game with victory points and ranking for each player.

At this point, clients are removed from the VirtualView. A client now can quit or play another game.