GRUPPO AM-09 – ASPESI, BETTINI, DE VITA

CLIENT SERVER COMMUNICATION PROTOCOL

The communication protocol we decided to implement is based on the serialization and exchange of java objects through socket object streams. The protocol is fully asynchronous. At each end of the connection, after an object is received, we use “instance of” calls to identify which class it belongs to.

We allow the creation of multiple matches, but we fill only one match at a time. When a match is created the server is ready to create another. Equal nicknames are forbidden in the same match but are allowed in different matches.

There are mainly two phases in the communication protocol:

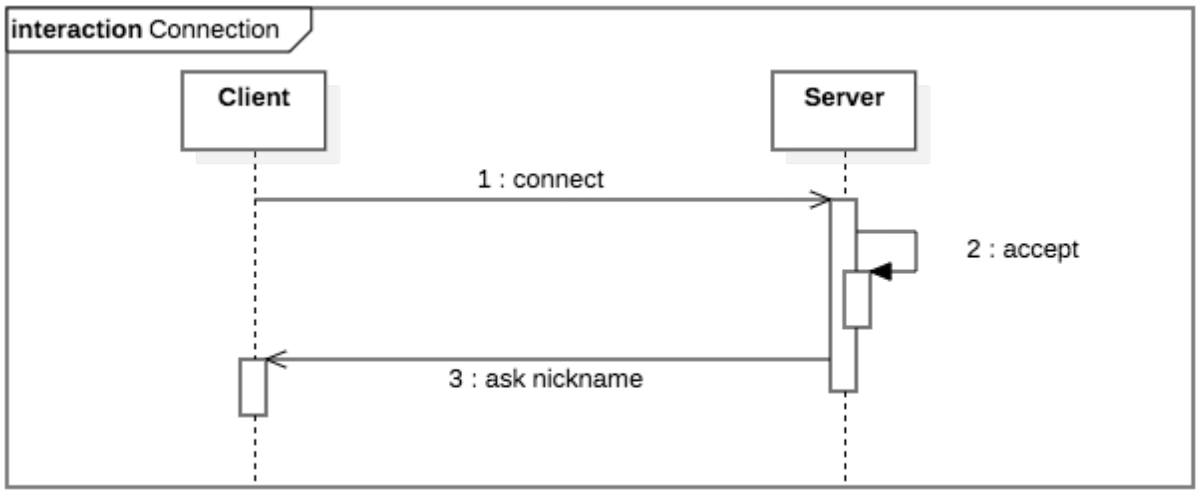
- The **setup/lobby phase** (which occurs every time a new client connects to the server)
- The **in-game phase** which starts immediately after the client is assigned to a match

Let’s explore them in detail.

SETUP PHASE

1 – Connection

The setup phase starts with a client connecting to the server. The client immediately receives a request to insert a nickname.



Ask nickname -> “Insert a nickname” [ConnectionMessages]

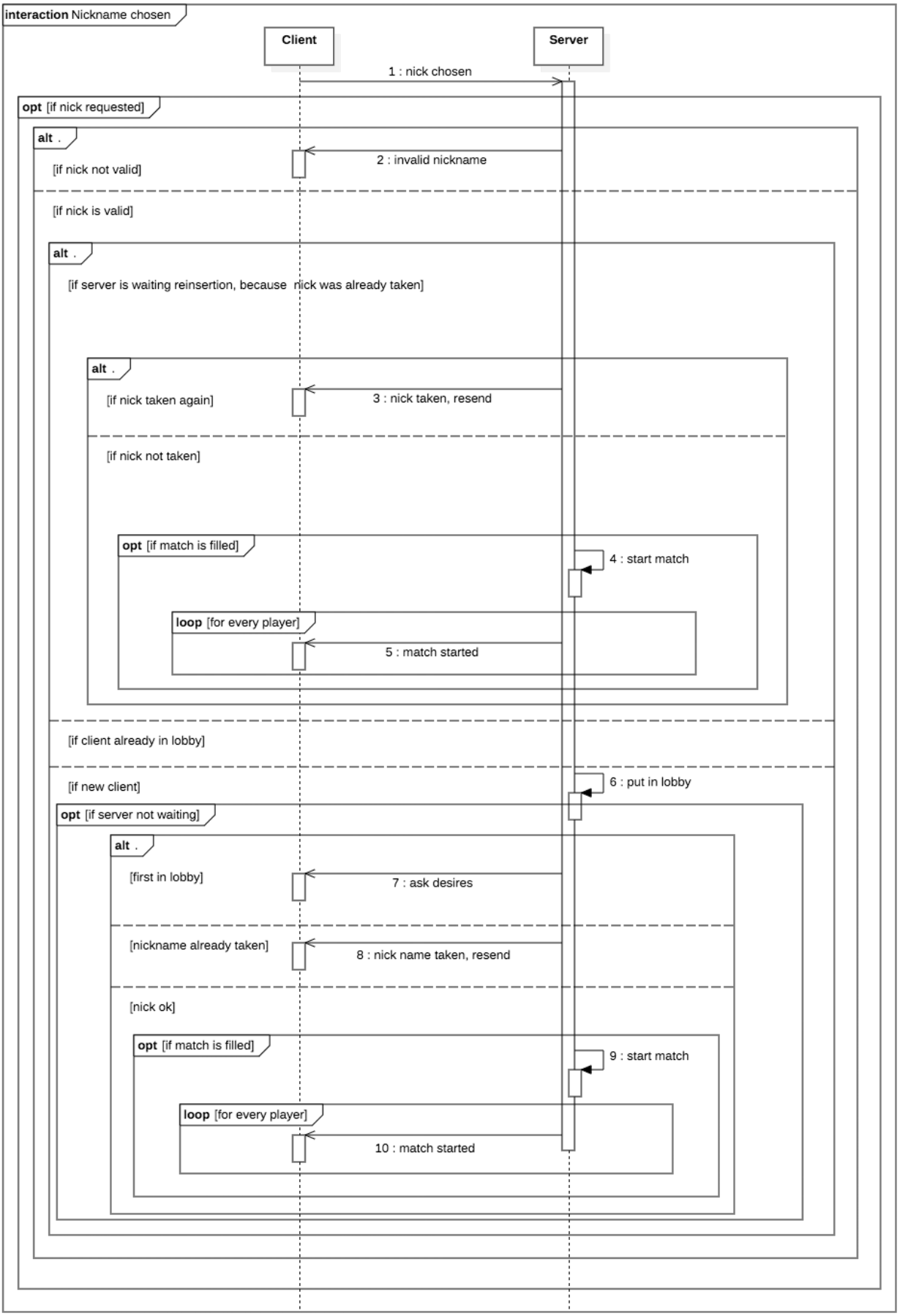
2 – Client sends nickname

The server receives a nickname. If the nick name has not been requested to this client the flow stops.
If the nick is invalid syntactically the server notifies the client and the flow stops.
Otherwise, we have 3 possibilities:

- The server state is “waiting for nick reinsertion because it was already taken” and it is waiting for this client. If the reinserted nick is “already taken” again, the server asks again for another. Otherwise, if in the lobby there are enough players to start the match, the match starts and all the participants are notified.
- The client is already in the lobby; the server ignores the client
- The client is a new client:
  the server then puts the client in the lobby;
  if the server is currently waiting for a packet the flow stops.
  if the client is the first in the lobby, the server requests the desired number of players for the match and the desired game mode (normal or hardcore).
  Else, if the client is not the first in the lobby and the provided nickname has been already chosen by another player waiting, the server asks for another nickname.
  Otherwise the nickname is ok; if in the lobby there are enough players to start the match, the match starts and all the participants are notified.

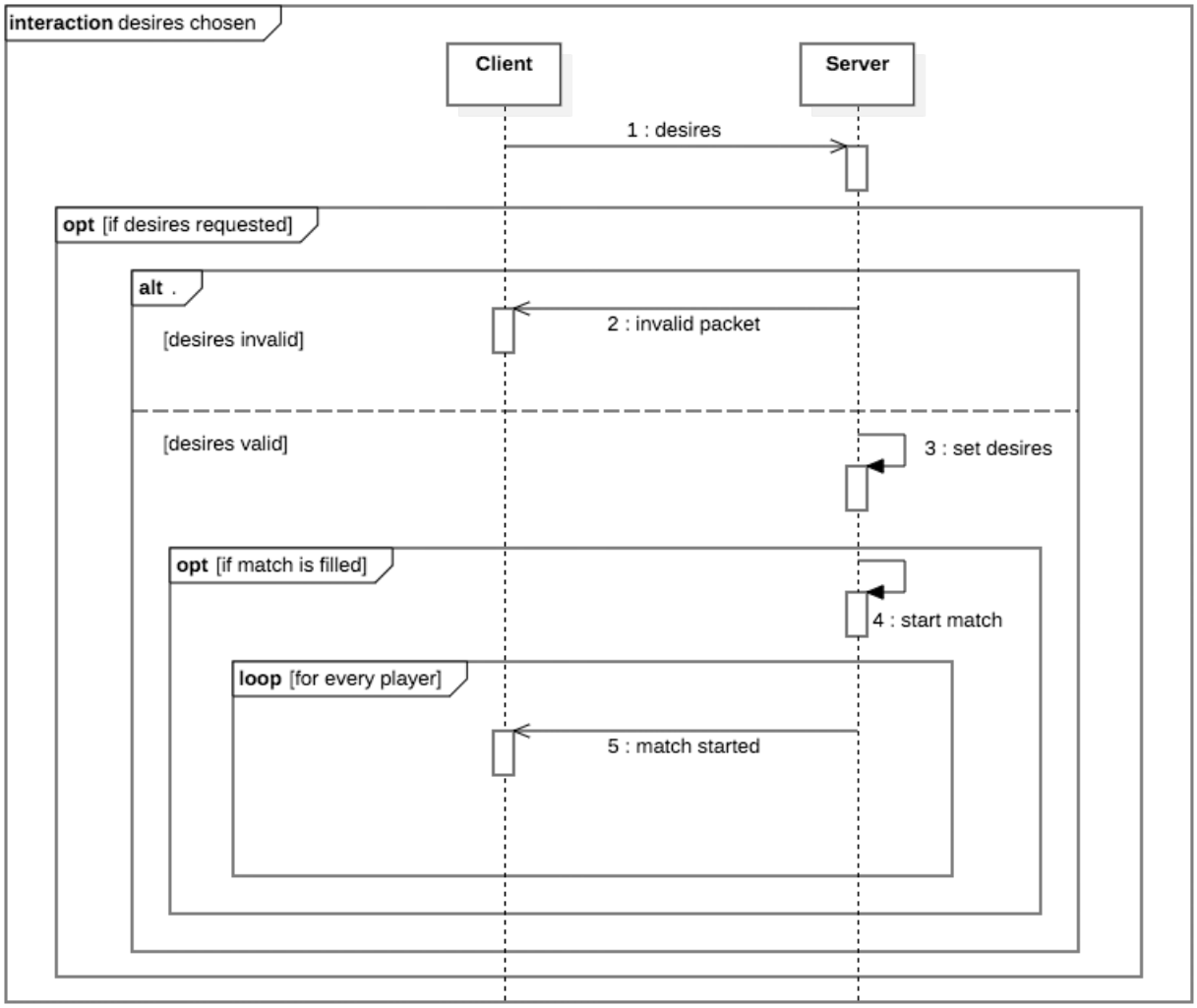
Sequence diagram next page...

Invalid nickname -> “The chosen nickname is invalid” [ConnectionMessages]
Ask desires -> “Insert desired num of plyers and game mode” [ConnectionMessages]
Nickname taken -> “The selected nickname is already taken, resend” [ConnectionMessages]
Match started -> [list of players’ nicknames, game mode] [PacketMatchStarted]


interaction Nickname chosen
Client
Server
1 : nick chosen
opt [if nick requested]
alt .
[if nick not valid]
2 : invalid nickname
[if nick is valid]
alt .
[if server is waiting reinsertion, because nick was already taken]
alt .
[if nick taken again]
3 : nick taken, resend
[if nick not taken]
opt [if match is filled]
4 : start match
loop [for every player]
5 : match started
[if client already in lobby]
[if new client]
6 : put in lobby
opt [if server not waiting]
alt .
[first in lobby]
7 : ask desires
[nickname already taken]
8 : nick name taken, resend
[nick ok]
opt [if match is filled]
9 : start match
loop [for every player]
10 : match started

3 – Client sends number of players and game mode desires

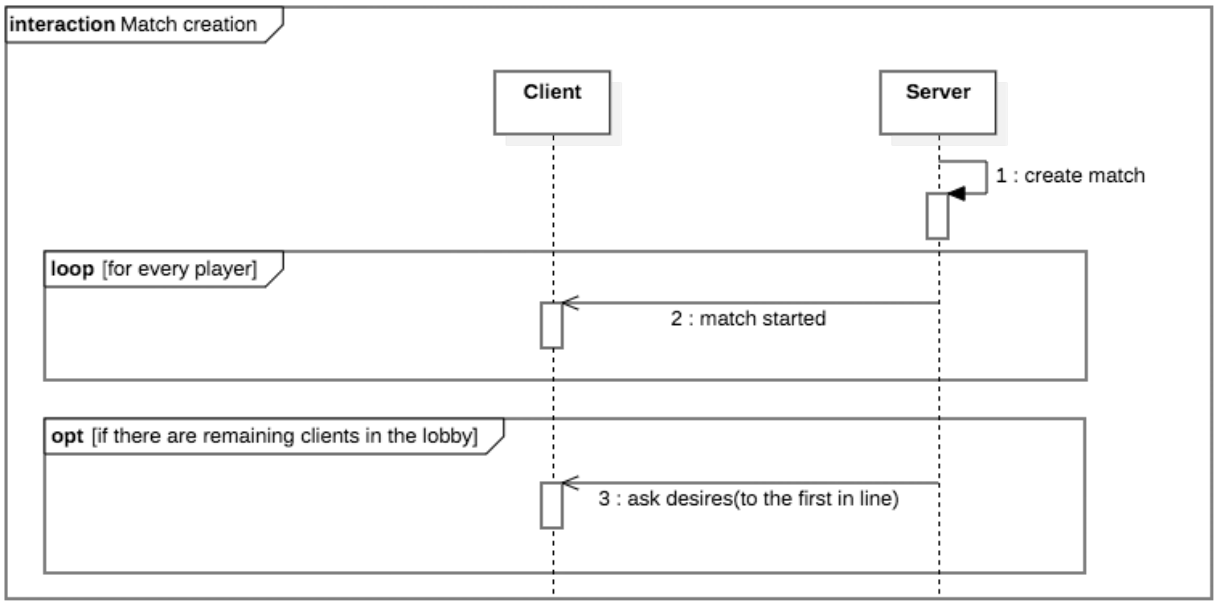
The server receives the desires. If the desires have not been requested to this client the flow stops.
If the desires are invalid the server notifies the client.
Otherwise, the current desired number of players and the current desired game mode are set.
If after this there are enough players in the lobby to start a match with those desires, the match is started.



Invalid packet -> "The packet sent is invalid" [ConnectionMessages]
Match started -> [list of players' nicknames, game mode] [PacketMatchStarted]

4 – Match creation

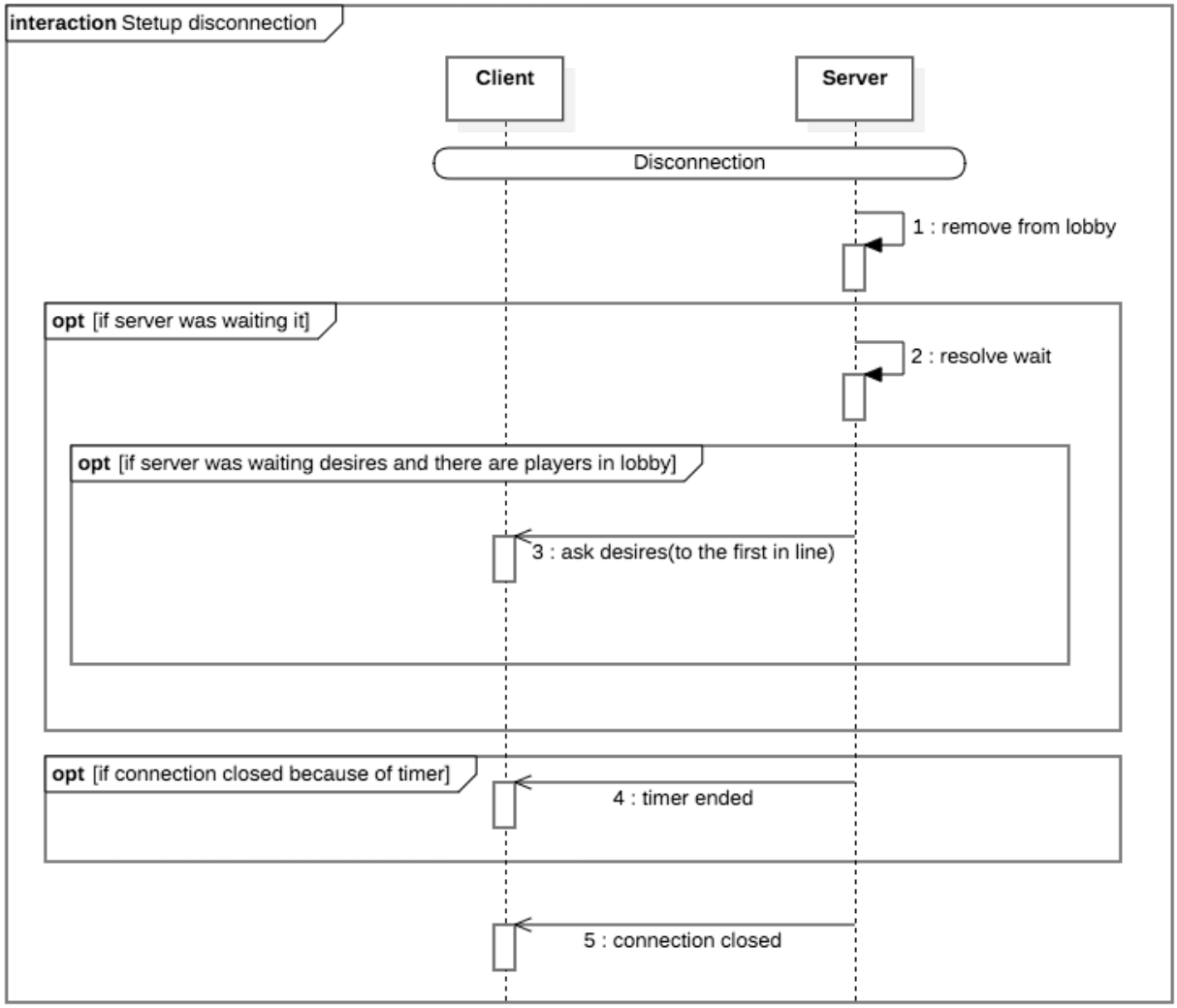
As seen above, match creation can occur in many situations when there are enough players to create the match (i.e. a waiting situation is resolved or simply enough clients connect).
When a match is created the participating players are notified and are removed from the lobby.
If after this, other clients remain in the lobby the server asks the desired number of players and game mode to the first in line.



Ask desires -> "Insert desired num of plyers and game mode" [ConnectionMessages]
Match started -> [list of players' nicknames, game mode] [PacketMatchStarted]

5 – Disconnection in setup phase

A client is considered disconnected when any type of error occurs when sending or receiving objects. Disconnection can also be forced by the server when a client takes too long to provide a requested packet (this is done by starting an asynchronous timer when sending request packets to the client).
When disconnection occurs during the setup phase the client is safely removed from the lobby.
If the server was waiting for a packet from this client the wait is resolved.
If the disconnection occurs because of the timer the server tries to notify it to the client
Finally the server tries to notify the complete closure of the connection and closes the socket.



Ask desires -> “Insert desired num of plyers and game mode” [ConnectionMessages]
Timer ended -> "You took too long to make your decision" [ConnectionMessages]
Connection closed-> "Connection closed" [ConnectionMessages]

IN-GAME PHASE

[rule 1]
As a premise, in all the sequence diagrams that will follow we suppose that when a client sends a packet, he has been requested to do so. The cases of non requested incoming packets to the server will result in said packets being ignored by the server itself.

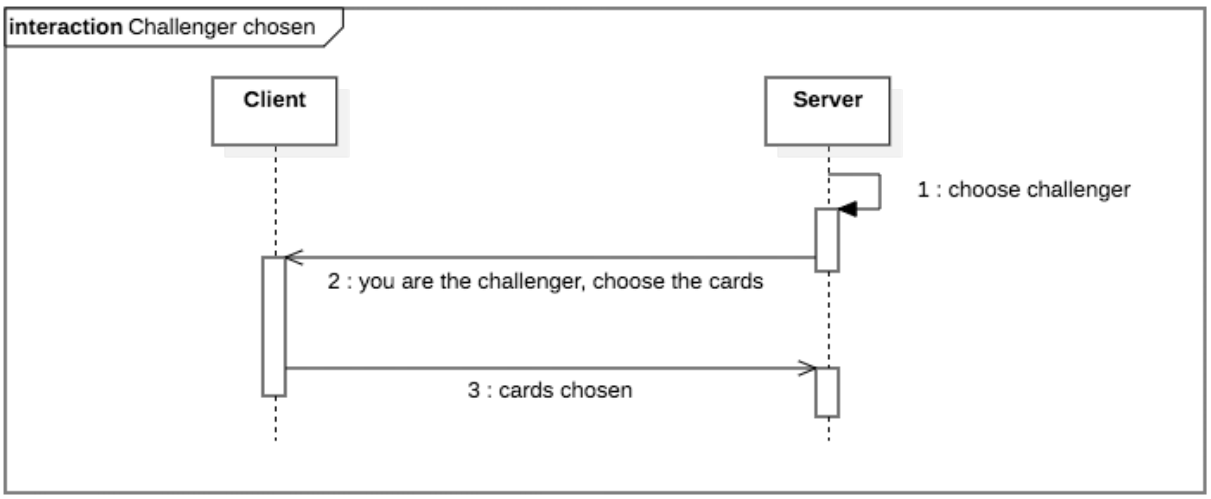
[rule 2]
If the client provides malformed or not valid information in a packet he has been requested, the server will notify him that the sent packet is invalid and he has to provide a valid packet of the type requested.

[rule 3]
Furthermore, when a server requests a packet from the client, a timer is always started; if the client takes too long to provide the requested answer disconnection will be forced by the server.
In-game disconnection will be treated deeply in paragraph **7**.

These behaviors hold during the whole in-game communication phase and are therefore omitted in the following.

1 – Challenger chosen

When a game starts, after all the players are notified, the server chooses a challenger and notifies him of all the available cards (with their descriptions) and of the number of cards that he has to choose.
The challenger then provides the selected cards.

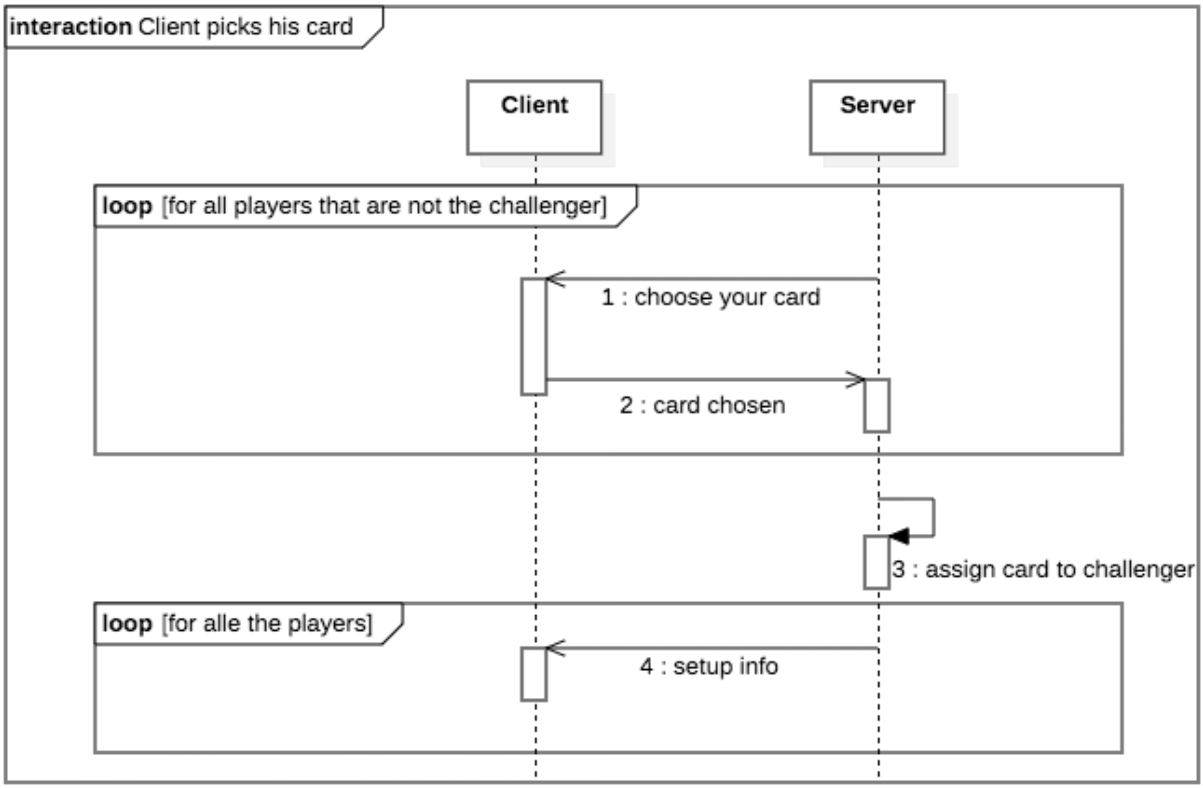


You are the challenger, choose the cards -> [recipient, number of cards to choose, all cards, available cards] [PacketCardsFromServer]
Cards chosen -> [chosen cards] [PacketCardsFromClient]

2 – Client picks his card

After the challenger has chosen the cards, one by one the other players are requested to choose a card from the ones selected by the challenger.
After they all select a card, the last card remaining is assigned to the challenger.
Then the server sends the setup information to all the players.
The setup information contains:

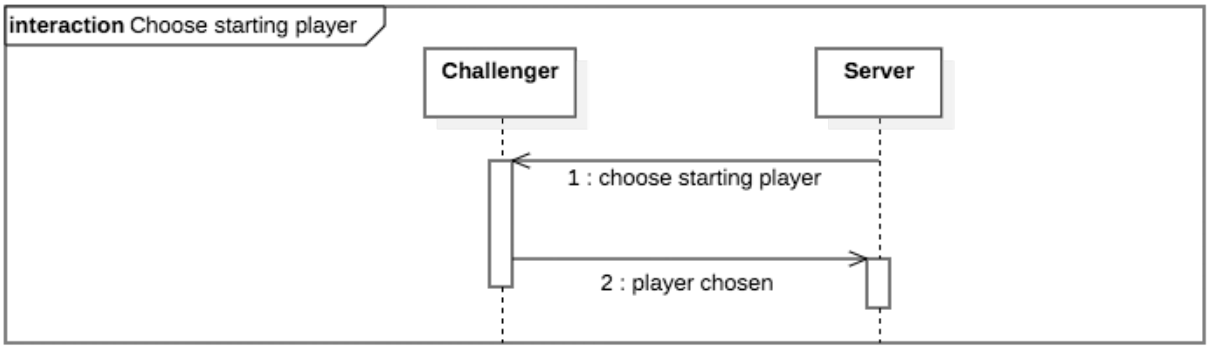
- The association between players and their cards
- The players' and workers' ids
- The association between players and their color



Choose your card -> [recipient, number of cards to choose (1), all cards, available cards] [PacketCardsFromServer]
Card chosen -> [chosen card] [PacketCardsFromClient]
Setup info -> [players and their cards, players' colors, players' and workers' ids] [PacketSetup]

3 – Challenger chooses start player

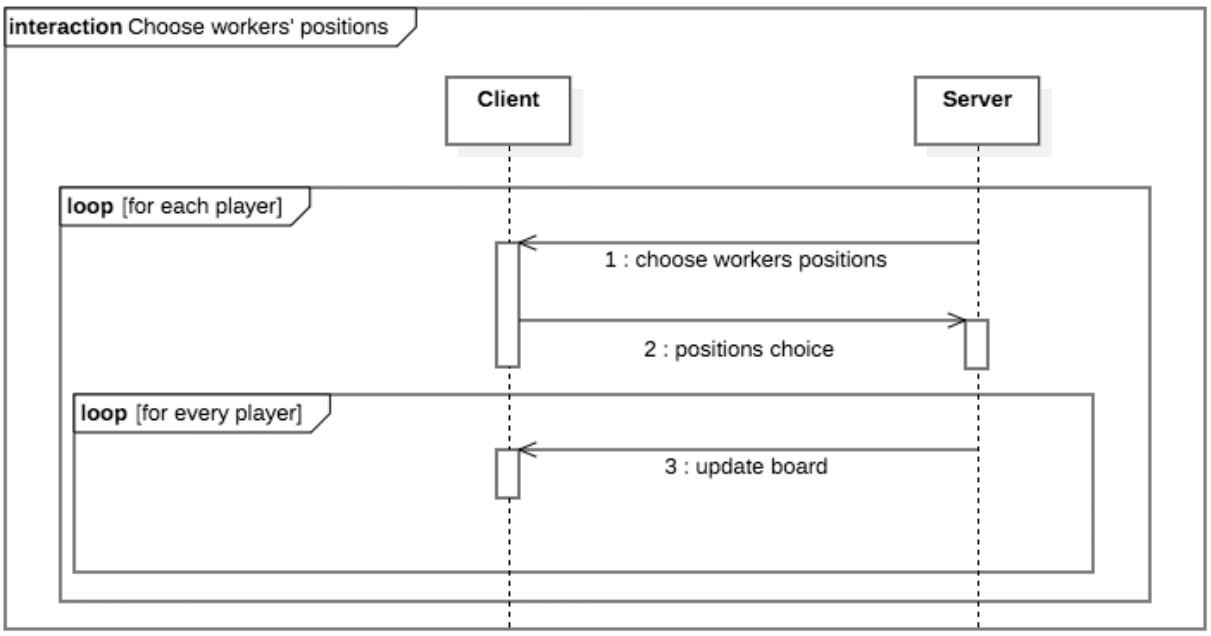
After the cards have been picked, the challenger is asked to pick a starting player.
Then he has to provide one.



Choose starting player -> [recipient, choose starting player] [PacketDoAction]
Player chosen -> [starting player] [PacketStartPlayer]

4 – Players choose workers' positions

One by one, beginning with the starting player, the server asks the clients to choose their workers' positions; each player makes his choice.
After the server receives a valid choice, a message with all the updated workers' positions on the board is sent to each player



Choose workers positions -> [recipient, choose workers positions] [PacketDoAction]
Positions choice -> [chosen positions (map id -> point)] [PacketWorkersPositions]
Update board -> [all the workers' positions selected till this moment] [PacketUpdateBoard]

5 – Move and build phases

The turn is divided in subsets which we can identify as **actions** (move, build).
A turn, thus, is comprised of a sequence of move and build actions.
The number of actions in the sequence and the order of said actions can depend on the god power of a player.

We implemented the possibility of playing in **two different game modes**: hardcore and normal.
In the **normal game mode** a player cannot make a wrong move or build and, therefore, is guided through the turn with a series of possible choices given by the server (which will be highlighted in the UI).
In the **hardcore game mode** a player does not receive any suggestion on the possible choices he has, and, when he makes a wrong move, two things could happen:

- he violates the basic game rules: in this case he is asked to repeat the last action (the packet is considered invalid **[rule 2]** applies)
- he violates other players' god powers and, thus, immediately loses the game

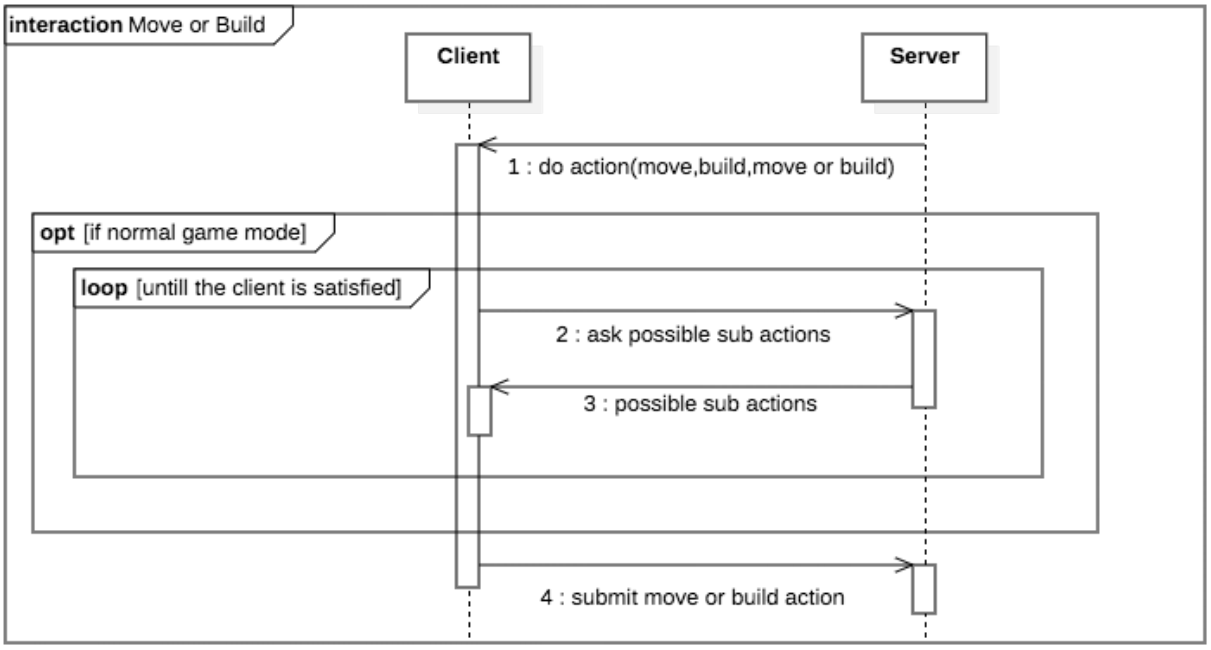
During a move or build action in normal mode a client has to ask multiple times to the server a list of the possible sub actions, this is because some gods allow move or build phases which are comprised of more than one inner action (Artemis(moves twice), Demeter(builds twice), etch...) and we aim to suggest to the client all the possible inner actions one by one.

The move and build actions in hardcore mode, communication wise, are a simplified version of the same actions in normal mode; as they do not require the continuous interrogation of the server to acquire the possible inner actions.

Lastly, build and move action are totally alike so we will get in depth with a sequence diagram representative for both of them.

The protocol

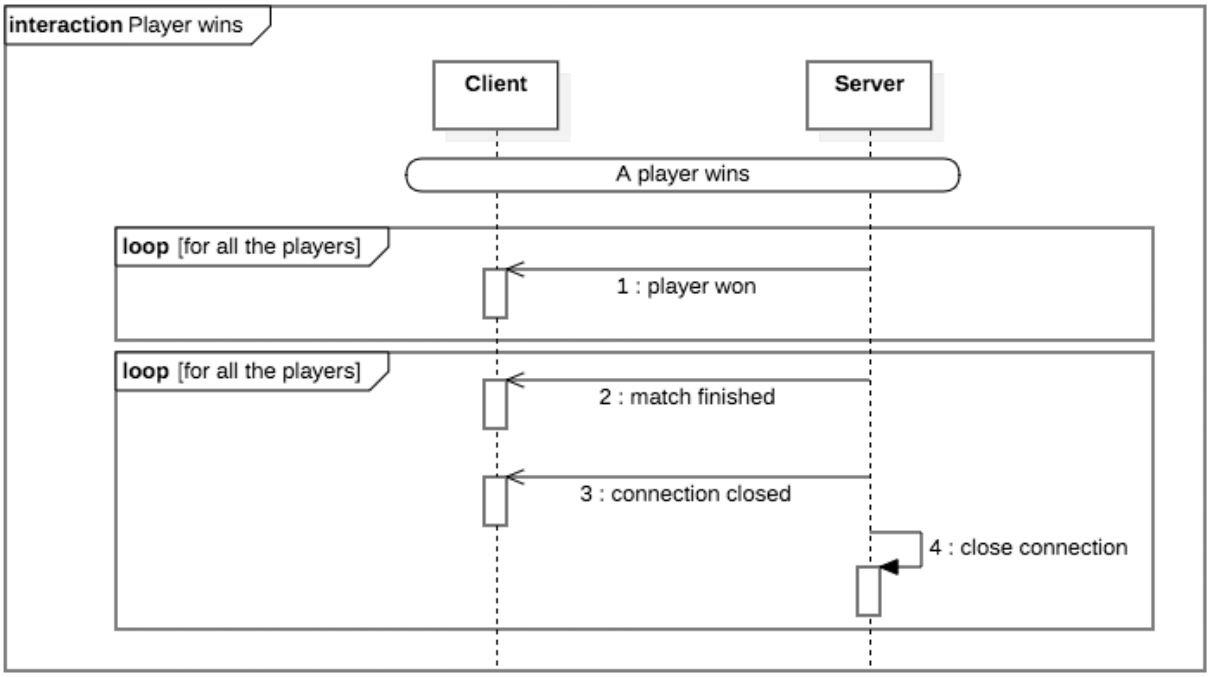
When it is a client's turn, the server asks the client to perform an action.
The action can be (move, build, move or build [this means that the client can choose which one he wants to perform e.g. Prometheus]).
In normal game mode the client can now query the server as many times as he wants to acquire information on the allowed inner actions that the player can perform.
Once the client is satisfied with his action, he can submit the entire action to the server which will act according to **[rule 2].**
Note that the loop in this diagram is not infinite as **[rule 3]** applies.



Do action (move, build, move or build) -> [recipient, action type] [PacketDoAction]
Ask possible sub actions -> [player's name requesting, for which worker he is requesting (optional), which sub actions the player has already planned for this worker(optional)] [PacketMove/PacketBuild]
Possible sub actions -> [the set of all possible sub actions relatively to the context of the client question] [PacketPossibleMoves/PacketPossibleBuilds]
Submit move or build action -> [the player doing the action, the worker involved, the set of sub action comprising the action] PacketMove/PacketBuild]

6 – Player won/lost

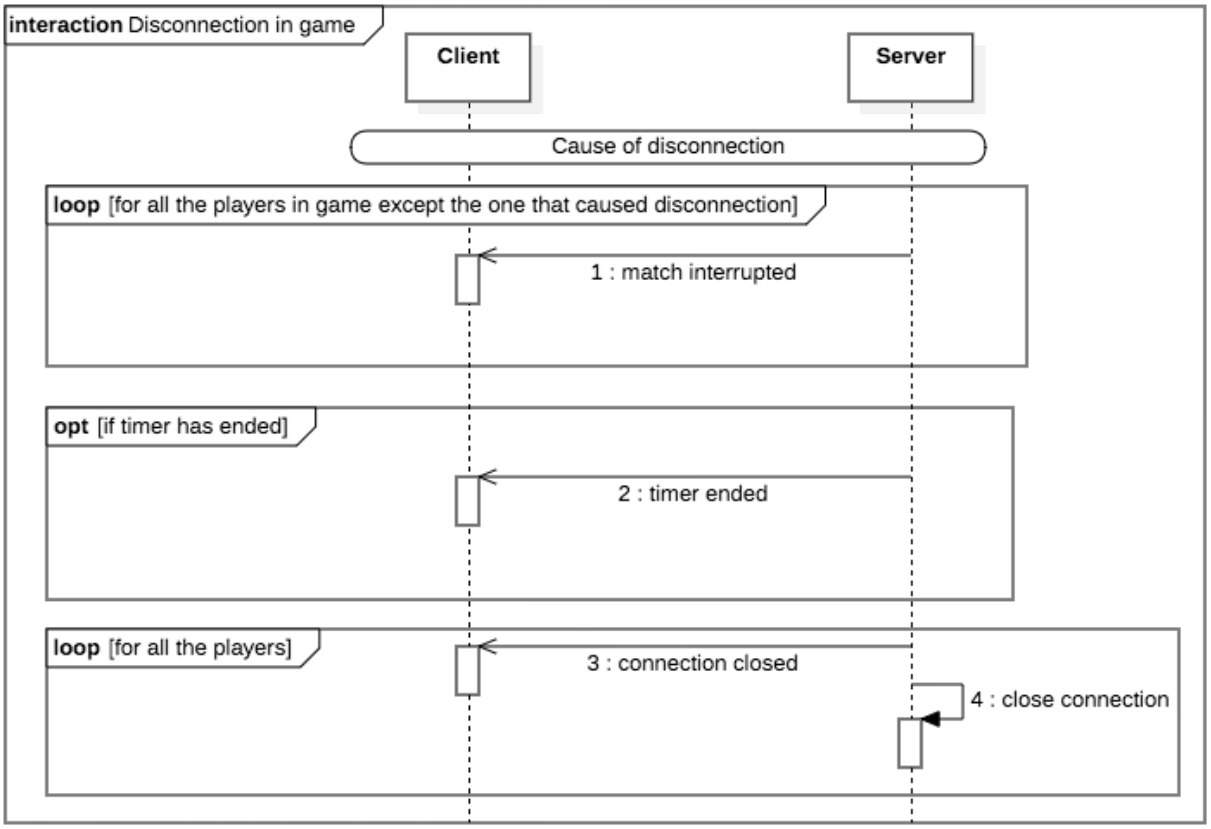
When a player wins or loses (unable to move or build | loss in hardcore) the server notifies all the other players. In case of a loss, if at least two players remain in the match, the match is continued as normal. Otherwise, if only one player remains, he wins the game.
Therefore, when a match ends not due to disconnections, there is always a winner.
The server notifies the players of the win situation.
Then sends a message of game finished to all the clients.
Then notifies the connection closure and closes the connection.



Player won -> [winner's name] [PacketUpdateBoard]
Match finished -> " A player has won the match and thus the game is finished" [ConnectionMessages]
Connection closed-> "Connection closed" [ConnectionMessages]

7 – Disconnection in game

A client is considered disconnected when any type of error occurs when sending or receiving objects. Disconnection can also be forced by the server when a client takes too long to provide a requested packet **[rule 3]**.
When a player disconnects from a match the match is ended.
The server tries to notify the player which caused the match end of the reason (timer ended, errors in the connection).
Then tries to notify the other playes that the match has been forcefully ended due to disconnected or not responding clients.
Then closes all the players' connections.



Match interrupted -> “"Match ended due to disconnected or not responding clients” [ConnectionMessages]
Timer ended -> "You took too long to make your decision" [ConnectionMessages]
Connection closed-> "Connection closed" [ConnectionMessages]