Network protocol presentation document
Peer review #2

Berardinelli, Genovese, Grandi, Haddou
Gruppo AM21

April 30th, 2024

1 Network protocol

The network protocol we designed is meant to be implemented with both RMI and Client-Server Socket functionality. Both client and server are equipped with a message parser for serialized Java objects sent through the network and a set of RMI interfaces which are meant to update the views in the client and call controller methods to update the model in the server.

1.1 Notes on RMI

In this documentation only *socket* messages are represented, as there is duality in the two approaches since every message corresponds to a remote method invocation.

1.2 General message handling

1.2.1 Failed connection handling

After a connection is enstablished, if the servers fails to respond to a message before the timeout, the clients will try to resend the message for a maximum of 3 times. If the server still fails to respond, the client will close the connection and notify the user that the connection has been lost.

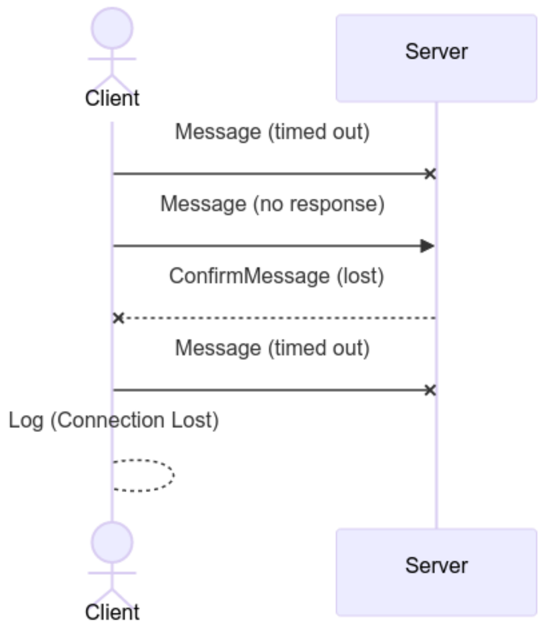


Figure 1: Connection Handling

1.2.2 "Not allowed" message handling

In the event a client sends a message for an action that the server doesn't expect or that they cannot perform in that moment, and in the event a client might be modified or 'enhanced' in a way the server does not contemplate, we have messages in place to send to the aforesaid client.

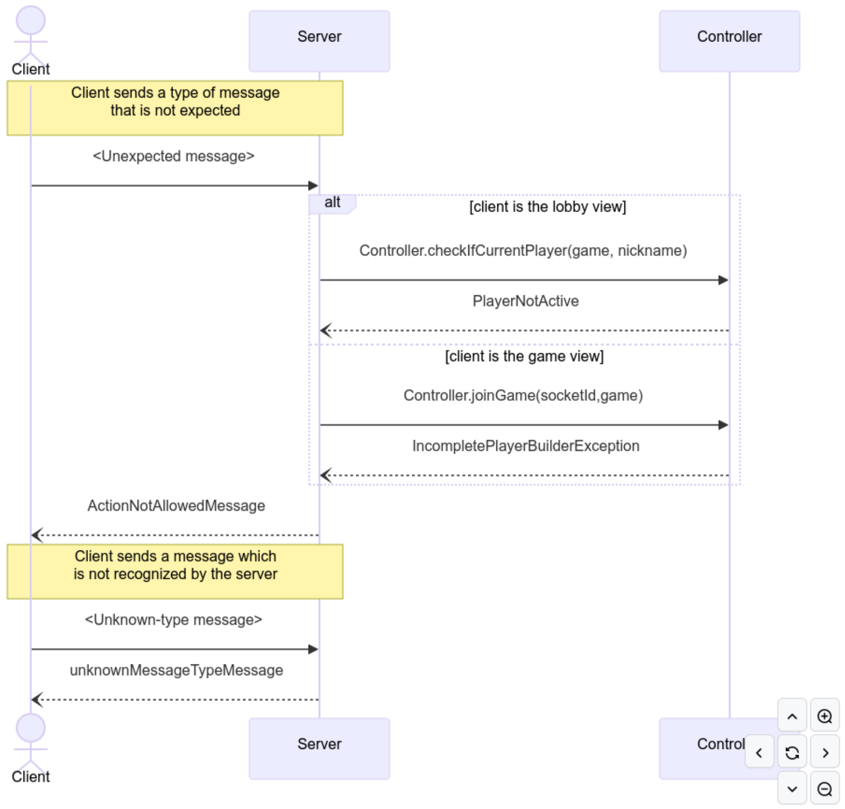

Client
Server
Controller
Client sends a type of message that is not expected
<Unexpected message>
alt
[client is the lobby view]
Controller.checkIfCurrentPlayer(game, nickname)
PlayerNotActive
[client is the game view]
Controller.joinGame(socketId,game)
IncompletePlayerBuilderException
ActionNotAllowedMessage
Client sends a message which is not recognized by the server
<Unknown-type message>
unknownMessageTypeMessage
Client
Server
Control

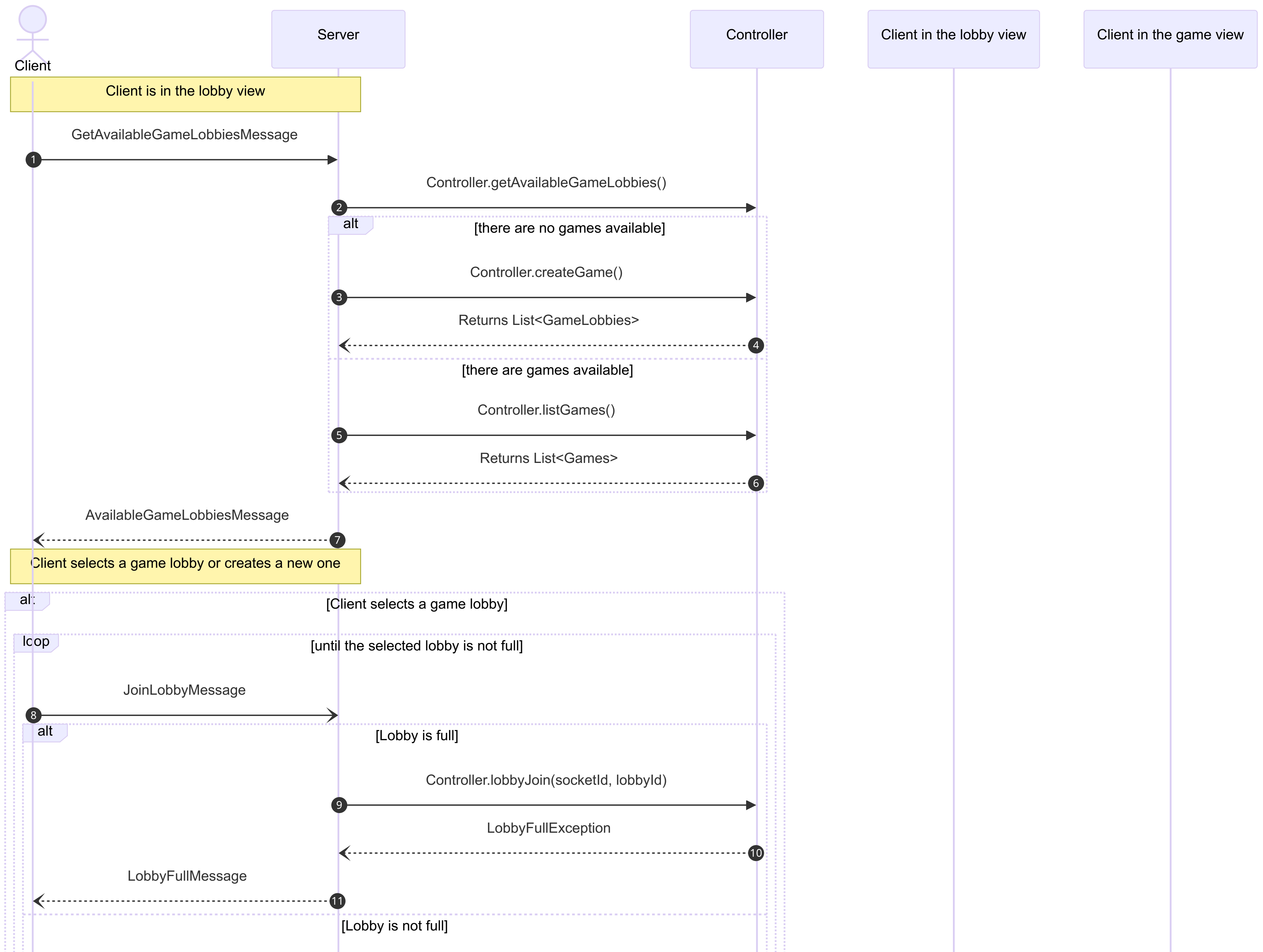
Figure 2: Connection Handling

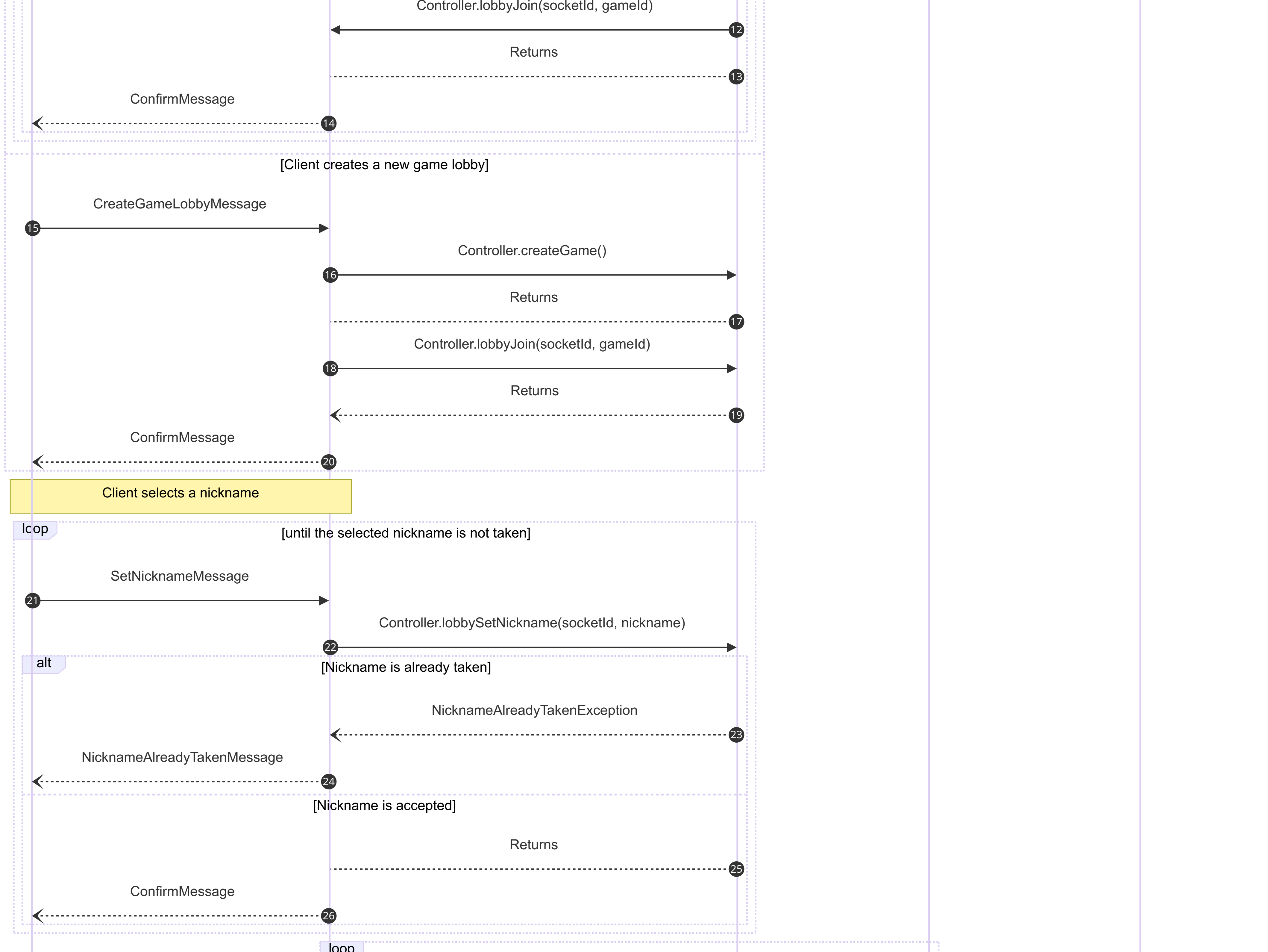
1.3 Game dynamics' flows

1.3.1 Lobby flow

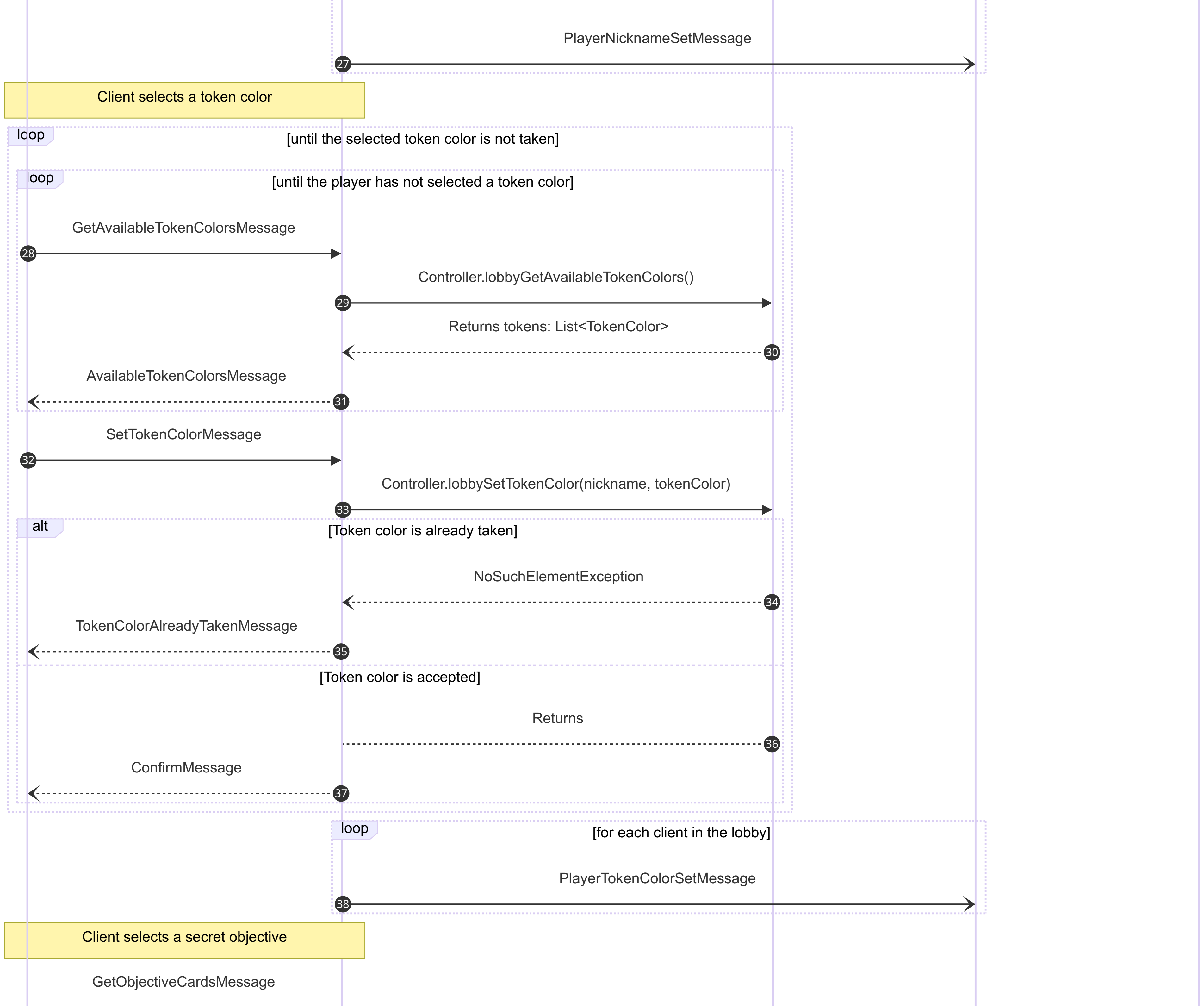
The player building process requires a series of essential steps, which are reported in the following sequence diagram.

Other than `ConfirmMessage`, which is required by the client to confirm the message has been received and handled correctly, we added a series of messages whose recipients are all the clients in the lobby or in the game. They are used to update the views of the clients and to notify them of the status of the lobby.

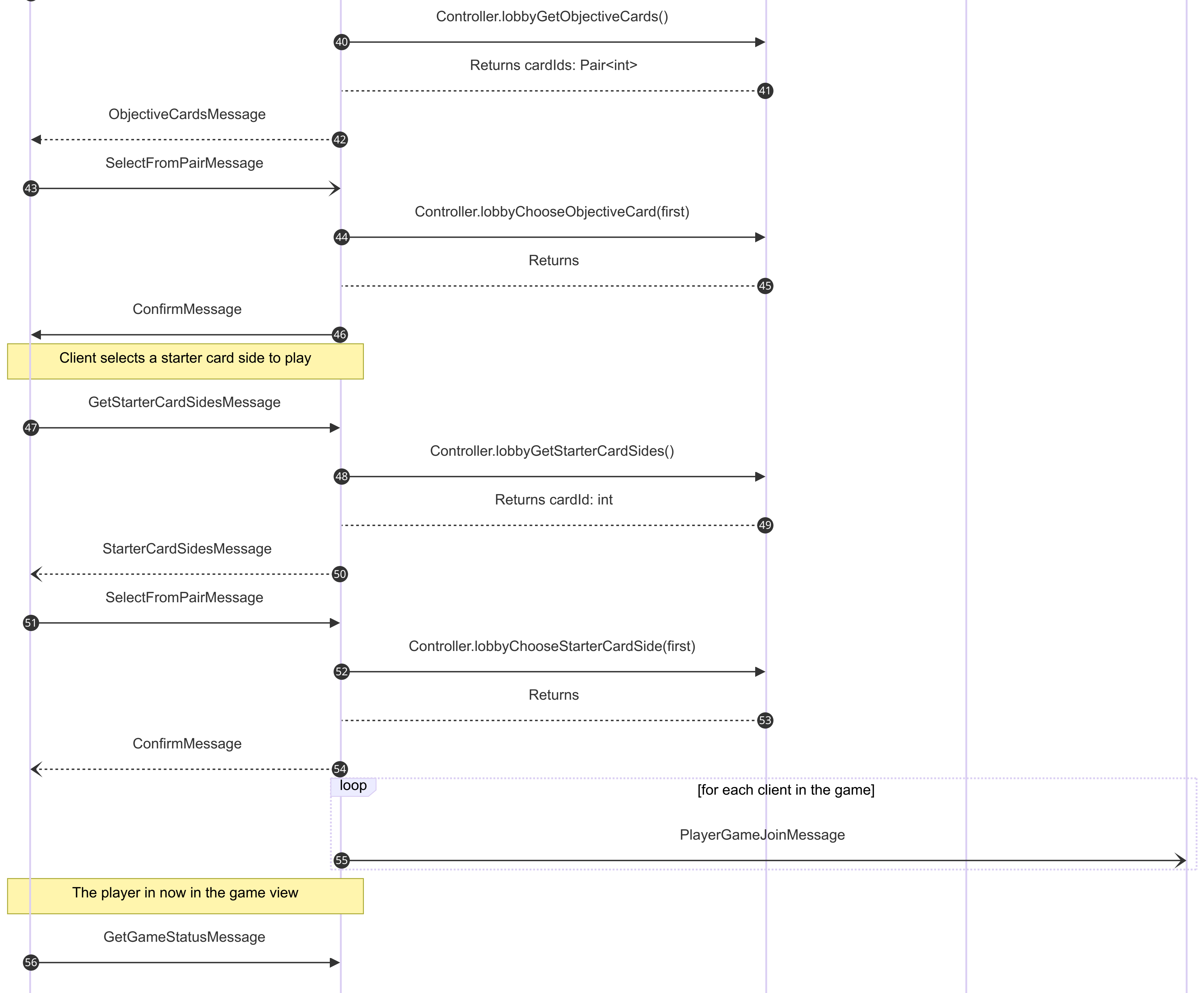

Client
Server
Controller
Client in the lobby view
Client in the game view
Client is in the lobby view
GetAvailableGameLobbiesMessage
1
Controller.getAvailableGameLobbies()
2
alt
[there are no games available]
Controller.createGame()
3
Returns List<GameLobbies>
4
[there are games available]
Controller.listGames()
5
Returns List<Games>
6
AvailableGameLobbiesMessage
7
Client selects a game lobby or creates a new one
alt
[Client selects a game lobby]
loop
[until the selected lobby is not full]
JoinLobbyMessage
8
alt
[Lobby is full]
Controller.lobbyJoin(socketId, lobbyId)
9
LobbyFullException
10
LobbyFullMessage
11
[Lobby is not full]

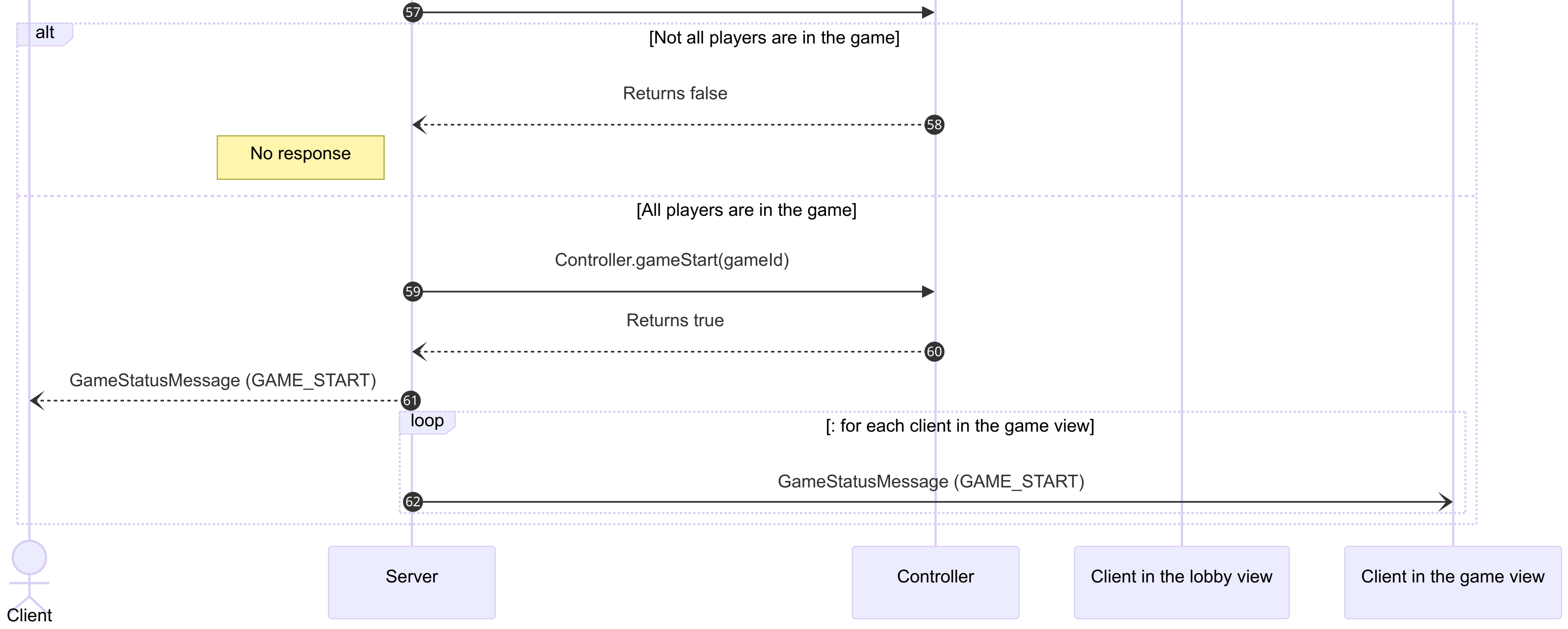


Controller.lobbyJoin(socketId, gameId)
12
Returns
13
ConfirmMessage
14
[Client creates a new game lobby]
CreateGameLobbyMessage
15
Controller.createGame()
16
Returns
17
Controller.lobbyJoin(socketId, gameId)
18
Returns
19
ConfirmMessage
20
Client selects a nickname
loop
[until the selected nickname is not taken]
SetNicknameMessage
21
Controller.lobbySetNickname(socketId, nickname)
22
alt
[Nickname is already taken]
NicknameAlreadyTakenException
23
NicknameAlreadyTakenMessage
24
[Nickname is accepted]
Returns
25
ConfirmMessage
26
loop



PlayerNicknameSetMessage
27
Client selects a token color
loop
[until the selected token color is not taken]
loop
[until the player has not selected a token color]
GetAvailableTokenColorsMessage
28
Controller.lobbyGetAvailableTokenColors()
29
Returns tokens: List<TokenColor>
30
AvailableTokenColorsMessage
31
SetTokenColorMessage
32
Controller.lobbySetTokenColor(nickname, tokenColor)
33
alt
[Token color is already taken]
NoSuchElementException
34
TokenColorAlreadyTakenMessage
35
[Token color is accepted]
Returns
36
ConfirmMessage
37
loop
[for each client in the lobby]
PlayerTokenColorSetMessage
38
Client selects a secret objective
GetObjectiveCardsMessage

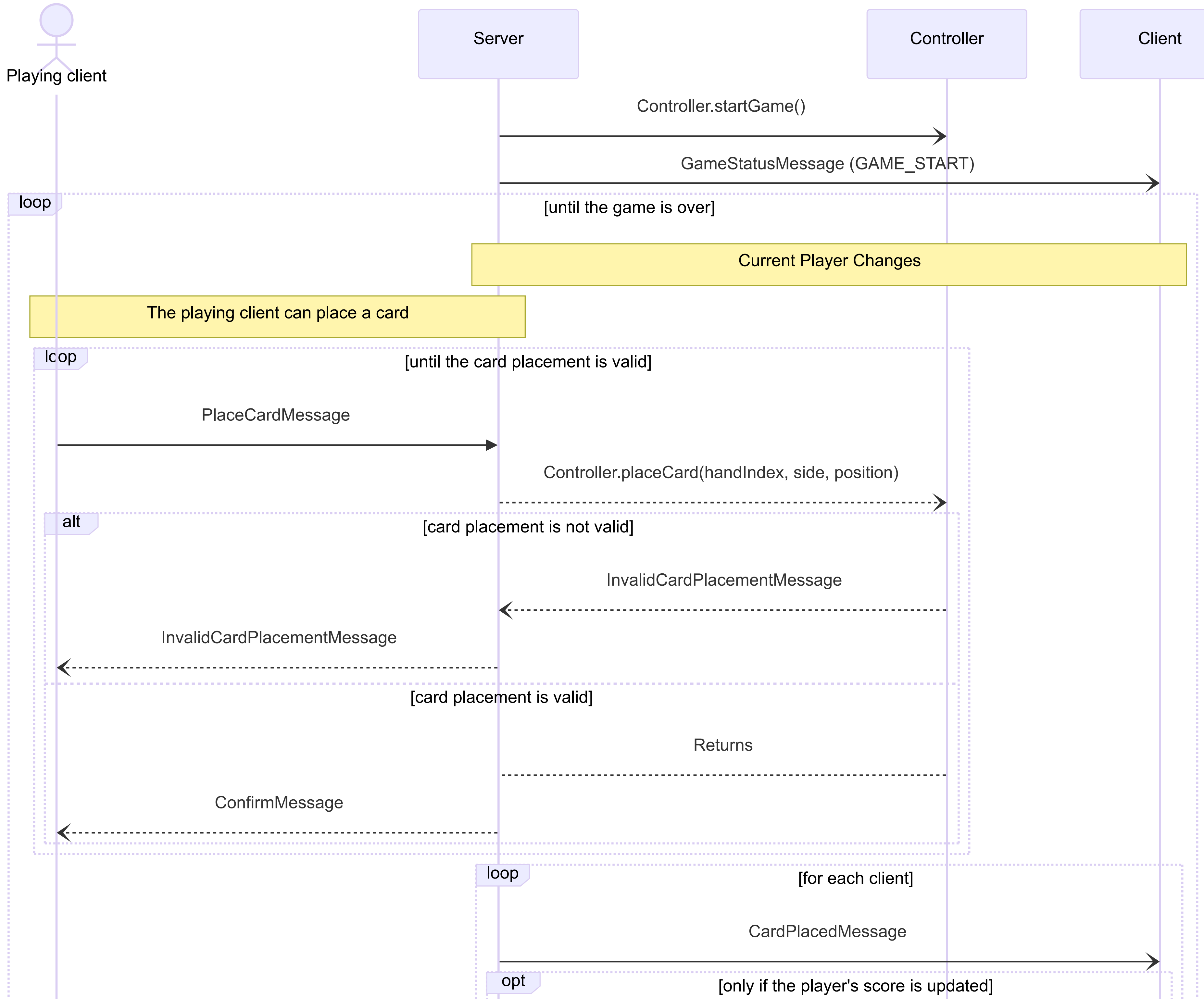

Controller.lobbyGetObjectiveCards()
40
Returns cardIds: Pair<int>
41
ObjectiveCardsMessage
42
SelectFromPairMessage
43
Controller.lobbyChooseObjectiveCard(first)
44
Returns
45
ConfirmMessage
46
Client selects a starter card side to play
GetStarterCardSidesMessage
47
Controller.lobbyGetStarterCardSides()
48
Returns cardId: int
49
StarterCardSidesMessage
50
SelectFromPairMessage
51
Controller.lobbyChooseStarterCardSide(first)
52
Returns
53
ConfirmMessage
54
loop
[for each client in the game]
PlayerGameJoinMessage
55
The player in now in the game view
GetGameStatusMessage
56

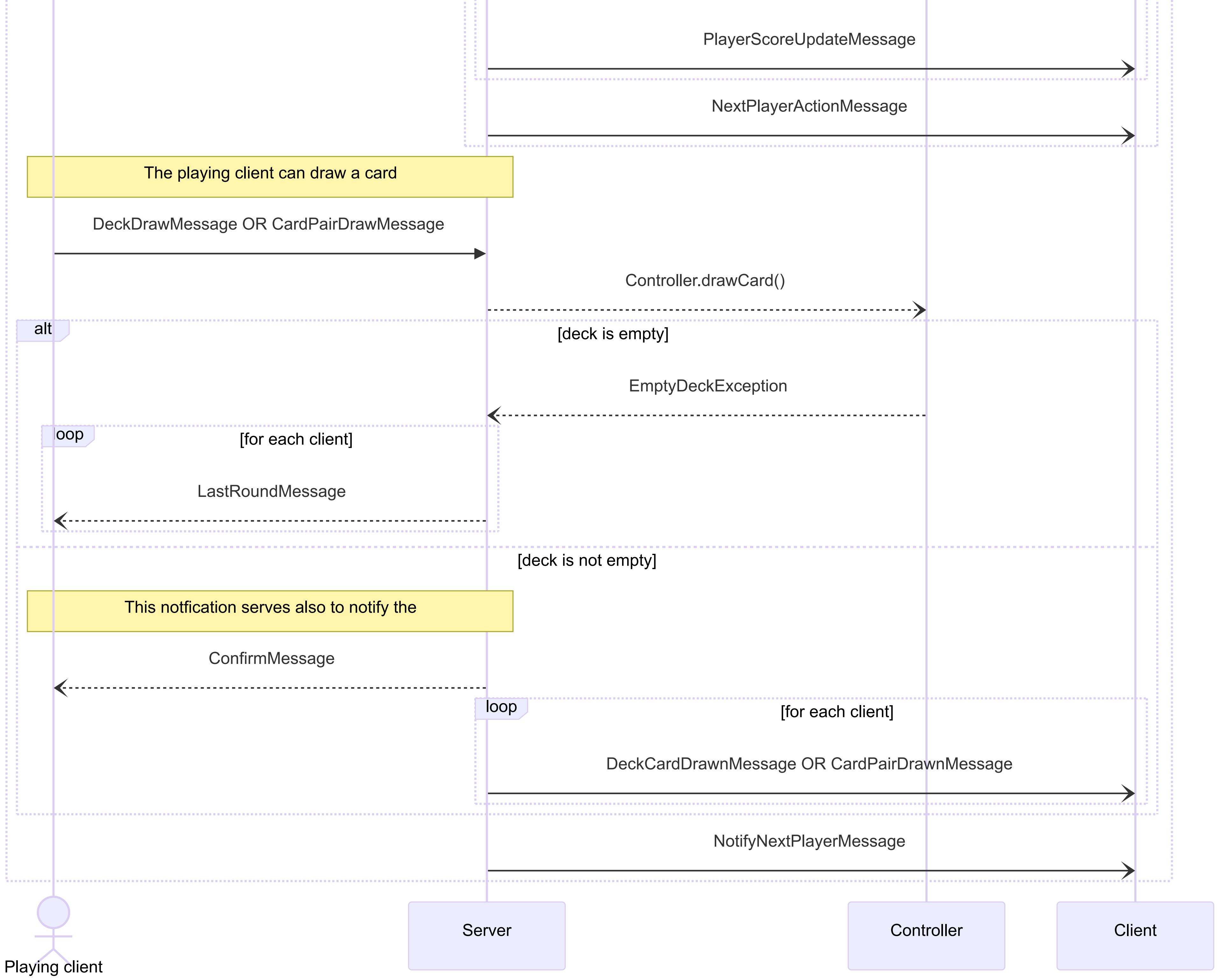

57
alt
[Not all players are in the game]
Returns false
58
No response
[All players are in the game]
Controller.gameStart(gameId)
59
Returns true
60
GameStatusMessage (GAME_START)
61
loop
[: for each client in the game view]
GameStatusMessage (GAME_START)
62
Client
Server
Controller
Client in the lobby view
Client in the game view

1.3.2 Normal game turns flow

Until `Game.nextTurn()` detects that a player has a winning score, the messages between the server and the clients are exchanged as follows.

As before, other than the `ConfirmMessage`, we have a series of messages whose recipients are all the clients in the game. They are used to update the views of the clients and to notify them of the status of the player turn.


Playing client
Server
Controller
Client
Controller.startGame()
GameStatusMessage (GAME_START)
loop
[until the game is over]
Current Player Changes
The playing client can place a card
loop
[until the card placement is valid]
PlaceCardMessage
Controller.placeCard(handIndex, side, position)
alt
[card placement is not valid]
InvalidCardPlacementMessage
InvalidCardPlacementMessage
[card placement is valid]
Returns
ConfirmMessage
loop
[for each client]
CardPlacedMessage
opt
[only if the player's score is updated]


PlayerScoreUpdateMessage
NextPlayerActionMessage
The playing client can draw a card
DeckDrawMessage OR CardPairDrawMessage
Controller.drawCard()
alt
[deck is empty]
EmptyDeckException
loop
[for each client]
LastRoundMessage
[deck is not empty]
This notfication serves also to notify the
ConfirmMessage
loop
[for each client]
DeckCardDrawnMessage OR CardPairDrawnMessage
NotifyNextPlayerMessage
Playing client
Server
Controller
Client

1.3.3 Game over flow

When `Game.nextTurn()` detects that a player has a winning score or an `EmptyDeckException` is caught by the controller, a message is sent to all the clients to notify them of the number of remaining rounds.

After the final rounds are played, the server will send a series of messages to all the clients to notify them that the game is over and update the final scores of the players after adding the objective cards' points.

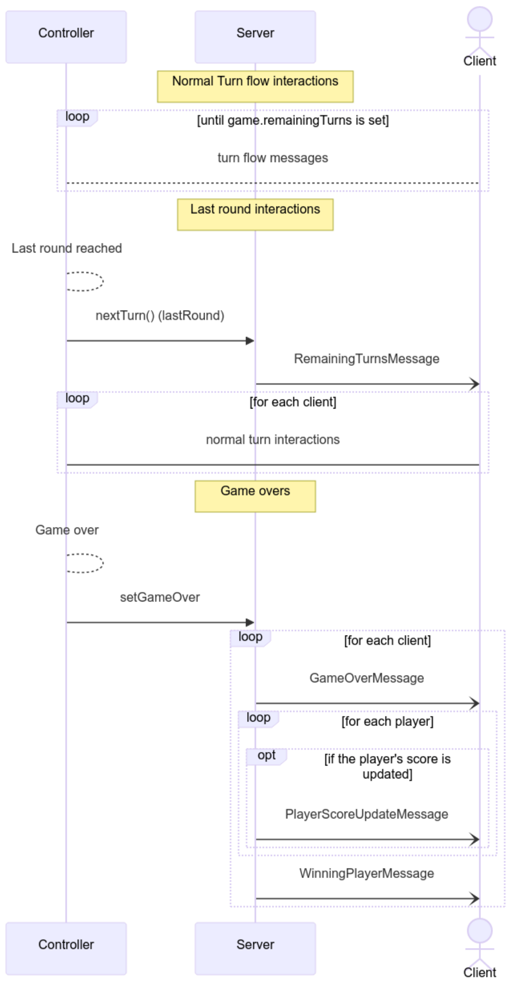


Figure 5: Gameover flow

1.4 Advanced Features flows

1.4.1 Chat

This exchange happens when a player (Client) wants to write a message in the chat. After PostMessage is sent, the server replies that the message has been received and posted. After that, the server sends a notification to all the recipients of the message informing them that there is a new message in the chat.

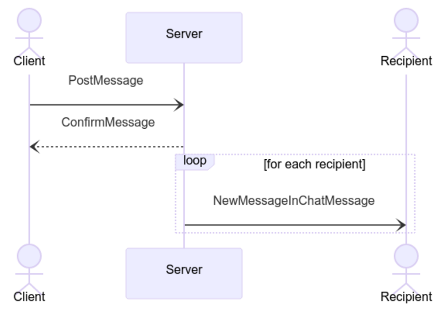


Figure 6: Chat flow

2 Message class structure and uml diagram

2.1 Message class structure

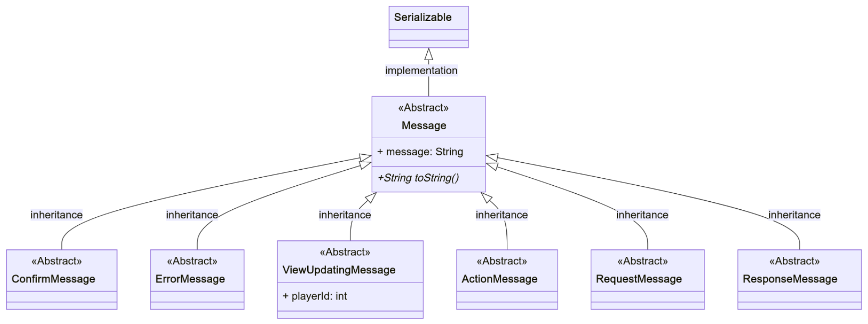


Figure 7: Message abstract class structure and hierarchy

2.2 Client requests

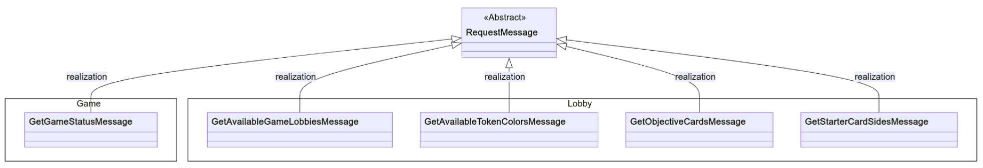


Figure 8: Client requests

2.3 Server responses

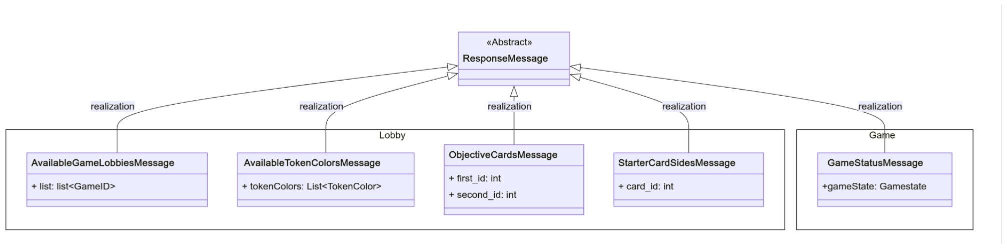


Figure 9: Server responses

2.4 Client actions

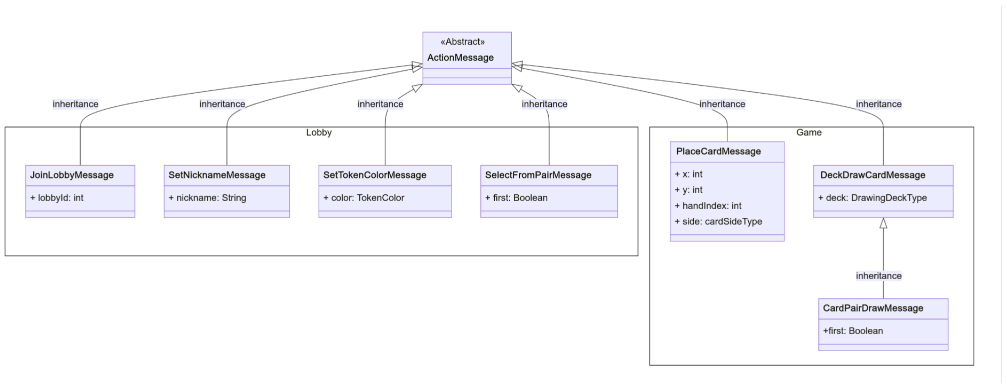


Figure 10: Client actions

2.5 View updates

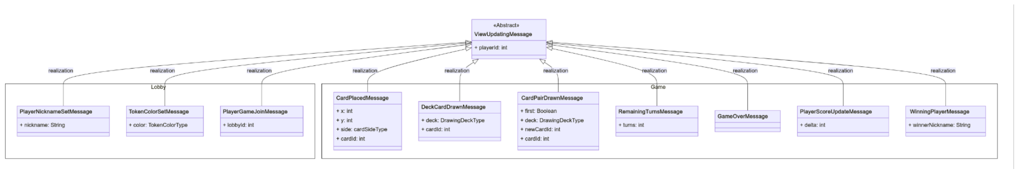


Figure 11: View updates

2.6 Server errors

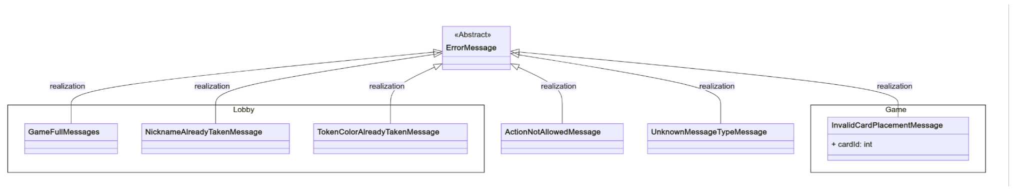

«Abstract»
ErrorMessage
realization
Lobby
GameFullMessages
NicknameAlreadyTakenMessage
TokenColorAlreadyTakenMessage
ActionNotAllowedMessage
UnknownMessageTypeMessage
Game
InvalidCardPlacementMessage
+ cardId: int

Figure 12: Server errors