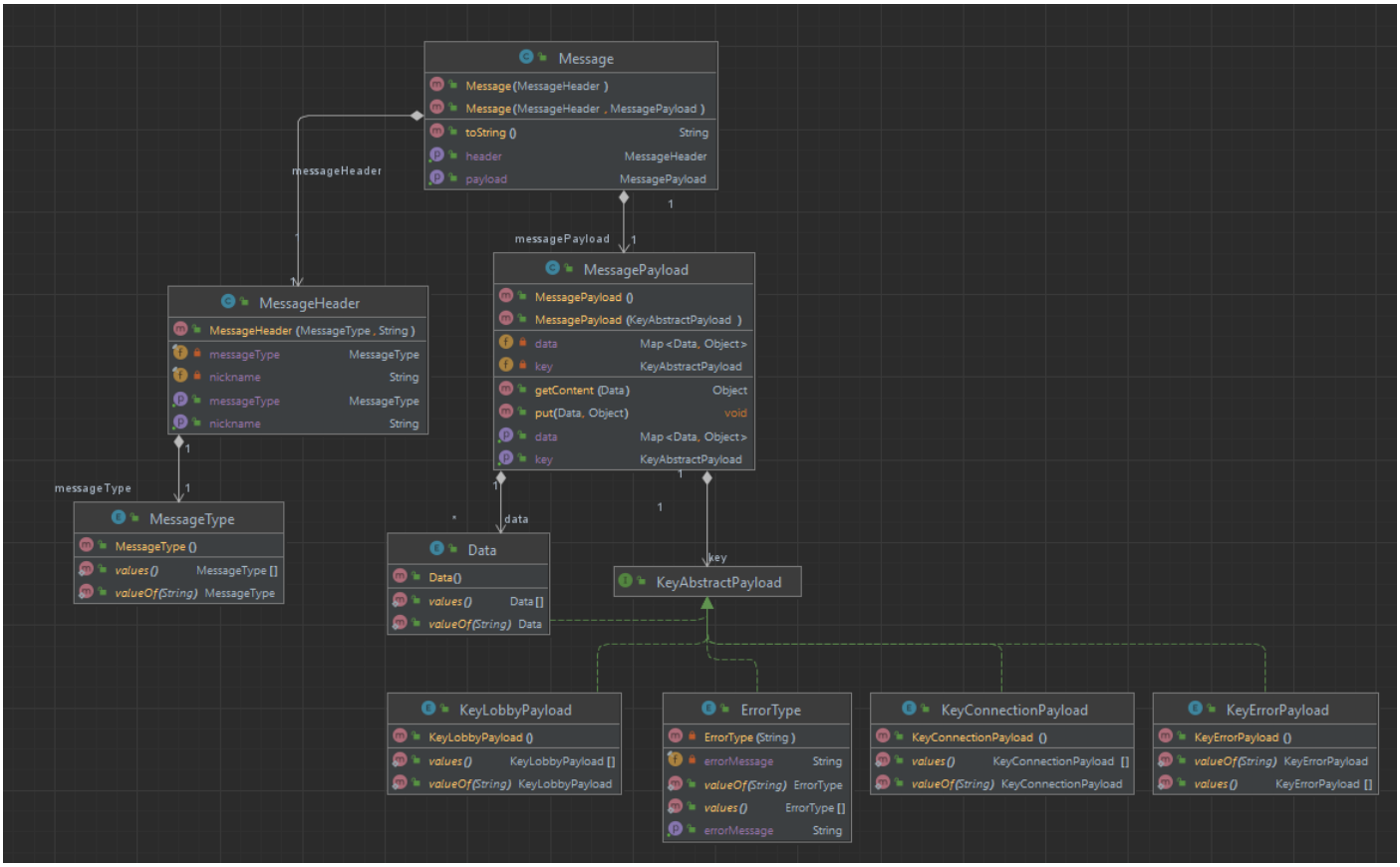
Communication Protocol GC41



1) MESSAGE

The Message class represents a message with a header and a payload. The header contains information about the type of the message and the nickname of the client who sent the message (or the client intended to receive the message if it is sent by the server). The payload contains a key (KeyAbstractPayload) that specifies the type of the message and a map of data (Map<Data, Object>) associated with their respective keys.

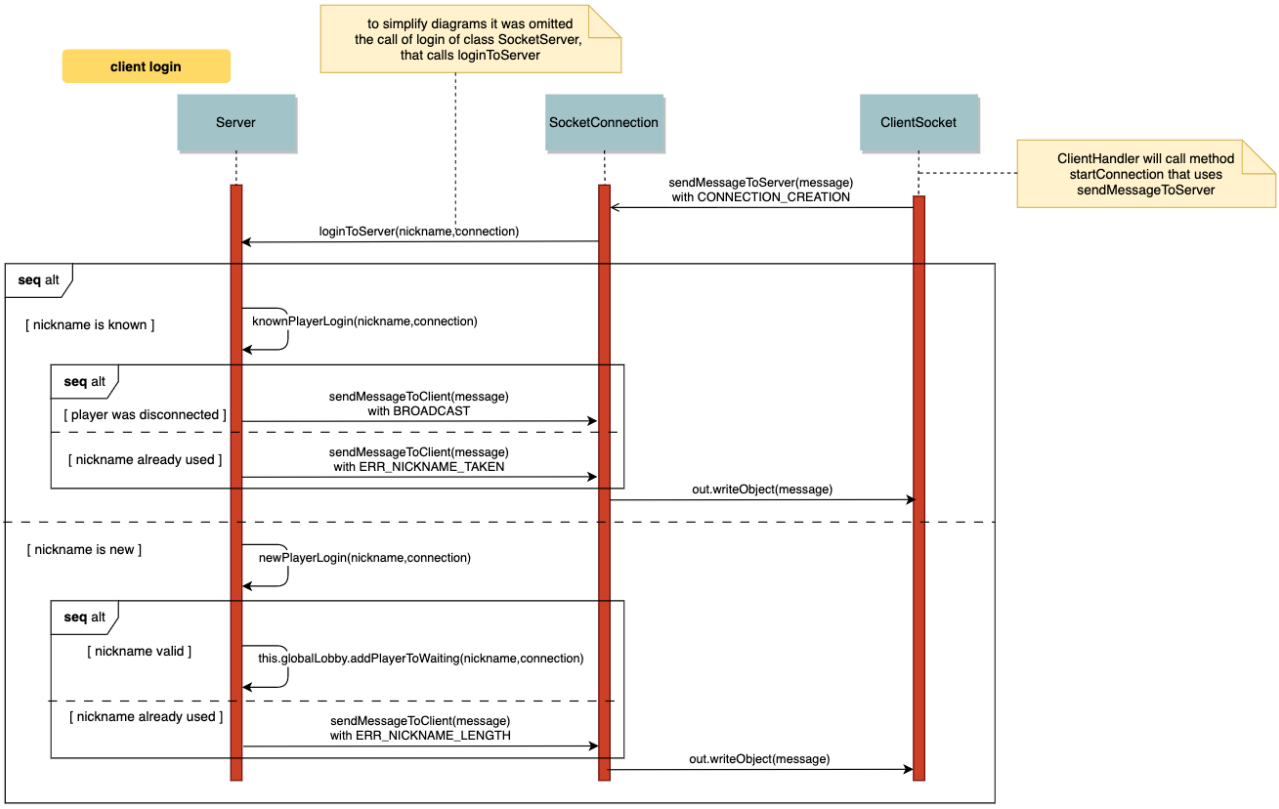
While messages from the server contain various data, messages from the client are structured in a consistent manner, utilizing "VALUE_CLIENT" as the sole payload key. The server determines its response to the message based on the "MessageType" and the “KeyAbstractPayload”.

Each potential error is accompanied by a specific string that is designed to be displayed to the client. This ensures that the client receives the relevant string corresponding to the specific error that occurred, allowing for clear and informative error messaging.

MESSAGE FROM CLIENT STRUCTURE

```
MessageHeader header=new MessageHeader(messageType, nickname);
MessagePayload payload=new MessagePayload(payloadKey);
payload.put(Data.VALUE_CLIENT,newValue);
Message message=new Message(header,payload);
```

2) LOGIN



a) CONNECTION MESSAGE *(server)*

```
MessageHeader header = new MessageHeader(MessageType.CONNECTION, getNickname());
MessagePayload payload = new
MessagePayload(KeyConnectionPayload.CONNECTION_CREATION);
```

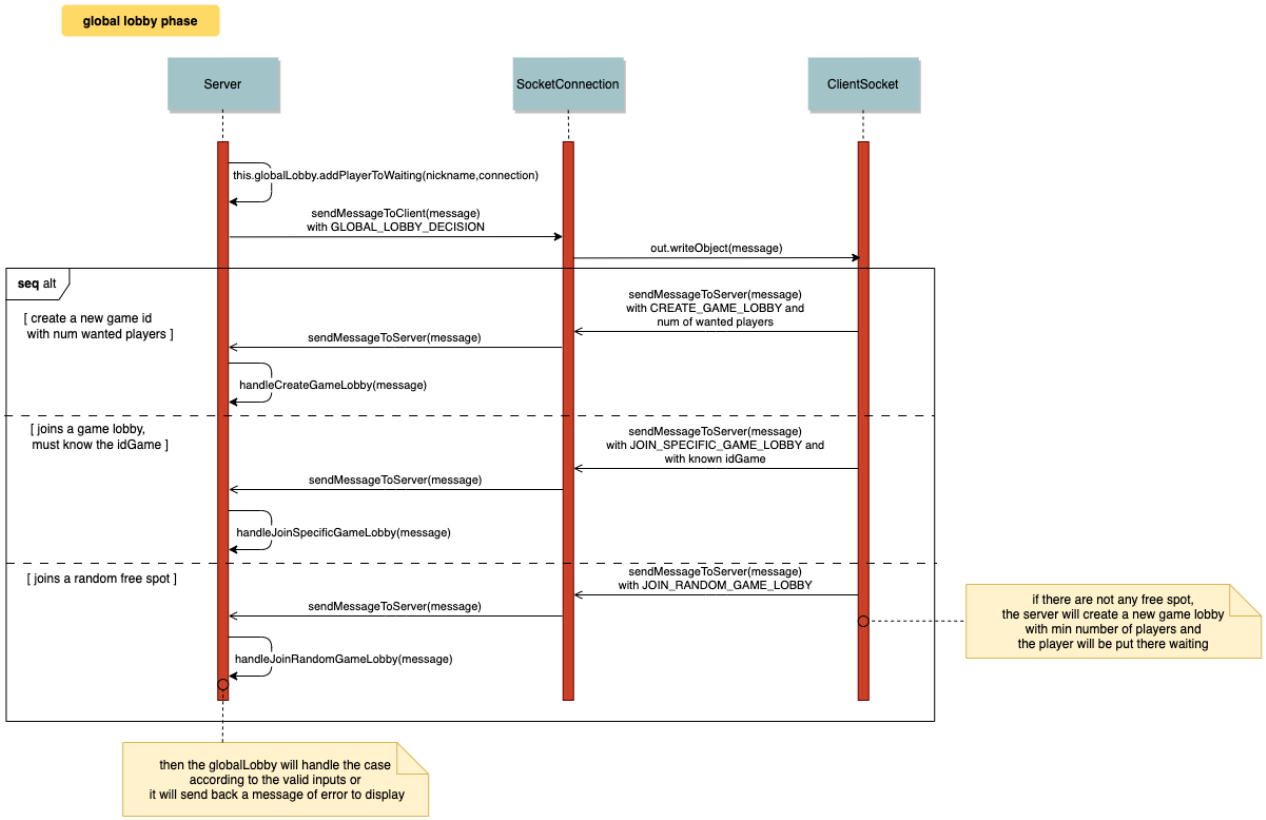
b) USER ERROR LOGIN *(server)*

```
MessageHeader header = new MessageHeader(MessageType.ERROR, nickname);
MessagePayload payload = new MessagePayload(KeyErrorPayload.ERROR_CONNECTION);
payload.put(Data.ERROR, ErrorType.ERR_NICKNAME_LENGTH)
Or
payload.put(Data.ERROR, ErrorType.ERR_NICKNAME_TAKEN);
```

Once the connection with the server is established, the server can respond with an error if the nickname provided does not meet certain length requirements or if it has already been taken by another user.

The server retrieves the specific length values from a JSON file named GameRules. This file contains parameters and settings for the game, including the minimum and maximum allowed length for nicknames. By modifying the corresponding values within the GameRules JSON file, the parameters can be easily adjusted.

3) LOBBY



a) LOBBY DECISION *(server)*

```
MessageHeader header = new MessageHeader(MessageType.LOBBY, nickname);
MessagePayload payload = new
MessagePayload(KeyLobbyPayload.GLOBAL_LOBBY_DECISION);
```

After successfully passing the nickname checks, the server sends a request to the client to choose the lobby option.

b) LOBBY DECISION *(client)*

MessageType-> *MessageType.*LOBBY;

The message is sent by the client after connecting to the server and logging in.

The three possible choices for the client are:

1) Create game lobby:
   KeyAbstractPayload-> KeyLobbyPayload.CREATE_GAME_LOBBY;
   VALUE_CLIENT-> number of players for the game that will be created.

2) Join specific game lobby:
   - *KeyAbstractPayload*-> KeyLobbyPayload.JOIN_SPECIFIC_GAME_LOBBY;
   - *VALUE_CLIENT*-> ID of the game lobby the client wants to join.

3) Join random game lobby:

- *KeyAbstractPayload*-> KeyLobbyPayload.JOIN_RANDOM_GAME_LOBBY.
  The client will be placed in the first available lobby that is waiting for the specified number of players indicated by the first connected client.

c) ERROR LOBBY *(server)*

```
MessageHeader header = new MessageHeader(MessageType.ERROR, nickname);
MessagePayload payload = new MessagePayload(KeyErrorPayload.ERROR_LOBBY);
```

Errors related to lobby operations:

1) Specified number of wanted players for creating a game is invalid *(CREATE NEW GAME LOBBY)*

```
payload.put(Data.ERROR, ErrorType.ERR_NUM_PLAYER_WANTED);
```

If the client enters an invalid number of players, the server will send this message indicating that the number of players is not acceptable. As a result, the client will be prompted to remake the lobby selection. The minimum and maximum number of players required to create a game are specified in a JSON file.

2) Lobby ID not found *(JOIN SPECIFIC GAME LOBBY)*

```
payload.put(Data.ERROR, ErrorType.ERR_GAME_NOT_FOUND);
or
payload.put(Data.ERROR, ErrorType.ERR_GAME_FULL);
```

If the client enters an invalid lobby ID that either does not exist or the game associated with the lobby has already started, the client will be prompted to make another lobby selection.

3) There are no available lobbies (JOIN RANDOM GAME LOBBY)

```
MessageHeader header = new MessageHeader(MessageType.CONNECTION,
nickname);
MessagePayload payload = new
MessagePayload(KeyConnectionPayload.BROADCAST);
payload.put(Data.CONTENT, ErrorType.ERR_NO_FREE_SPOTS.getErrorMessage());
```

If the client has chosen to "Join Random Game" and no lobby has been created yet, a lobby will be created with the minimum number of players allowed by the gameRules JSON file. In this case, the client will not be prompted to make the lobby selection again.

This message is structured differently from the other two.

4) GAME

a) START GAME OR RECONNECTION *(server)*

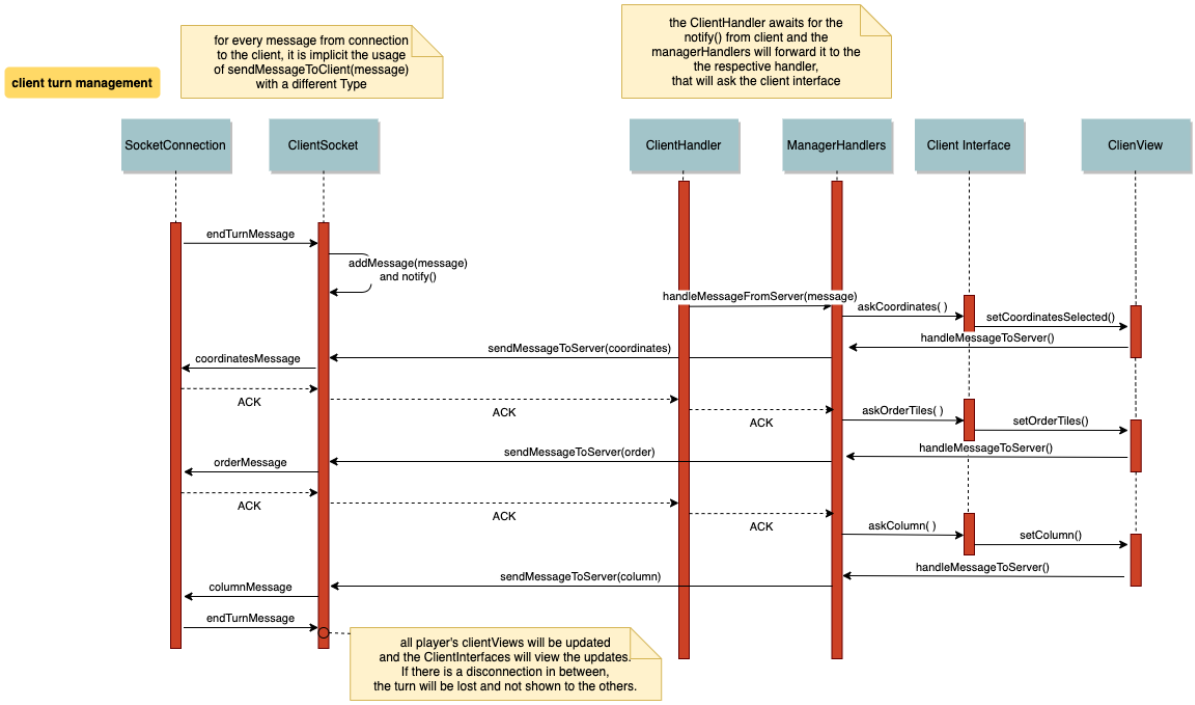
ALL_INFO message:

```
MessagePayload payload=new MessagePayload(TurnPhase.ALL_INFO);
BoardBoxView[][] boardView= modelView.getBoardView();
header=new MessageHeader(MessageType.DATA,nickname);
payload.put(Data.NEW_BOARD,boardView);
ItemTileView[][] bookshelfView=modelView.getBookshelfView(nickname);
payload.put(Data.NEW_BOOKSHELF,bookshelfView);
payload.put(Data.COMMON_GOAL,modelView.getCommonGoalView());
PersonalGoalCard personalGoalCard=modelView.getPlayerPersonalGoal(nickname);
payload.put(Data.PERSONAL_GOAL_CARD,personalGoalCard);
payload.put(Data.SELECTED_ITEMS,modelView.getSelectedItems());
payload.put(Data.MAX_SELECTABLE_TILES,modelView.getMAX_SELECTABLE_TILES());
payload.put(Data.PERSONAL_POINTS,modelView.getPersonalPoint(nickname));
payload.put(Data.POINTS,modelView.checkWinner());
payload.put(Data.NEXT_PLAYER,modelView.getTurnNickname());
payload.put(Data.PHASE,modelView.getTurnPhase());
```

Once all players have connected, the game will be created, and they will receive the "ALL_INFO" message. This message can also be sent after the game has started, in the event of a player reconnecting. This ensures that the player has all the necessary data to continue the game.

The value associated with the "NEXT_PLAYER" key in the message indicates the player who is currently required to take his or her turn. The message also includes information about the current "PHASE" of the game, which specifies the action the user needs to perform. Depending on the phase, the user will be prompted to do one of the following: select tiles from the board, arrange the order of tiles, or choose a column. The phase indicates the specific task or action that the user needs to complete.

b) PHASES OF THE TURN *(client)*



The messages related to the different phases contain an array in the "VALUE_CLIENT" field, while the "messageType" field indicates the type of data.

MessageType-> MessageType.DATA;

-SELECTION FROM BOARD:

- *KeyAbstractPayload->* TurnPhase.SELECT_FROM_BOARD*;*
- *VALUE_CLIENT->* coordinates of the chosen tiles. The even positions in the array represent the x-coordinates, while the odd positions represent the y-coordinates.

-ORDER TILES:

- *KeyAbstractPayload->* TurnPhase.SELECT_ORDER_TILES;
- VALUE_CLIENT-> An array that represents the order of the tiles would consist of values indicating the positions of the tiles in the desired sequence. For instance, consider an array with values {0, 2, 1}. This indicates the order chosen by the user for placing tiles in the bookshelf. In this example, the first tile to be placed in the bookshelf would be the tile at position 0, followed by the tile at position 2, and finally the tile at position 1.

-SELECT_COLUMN:

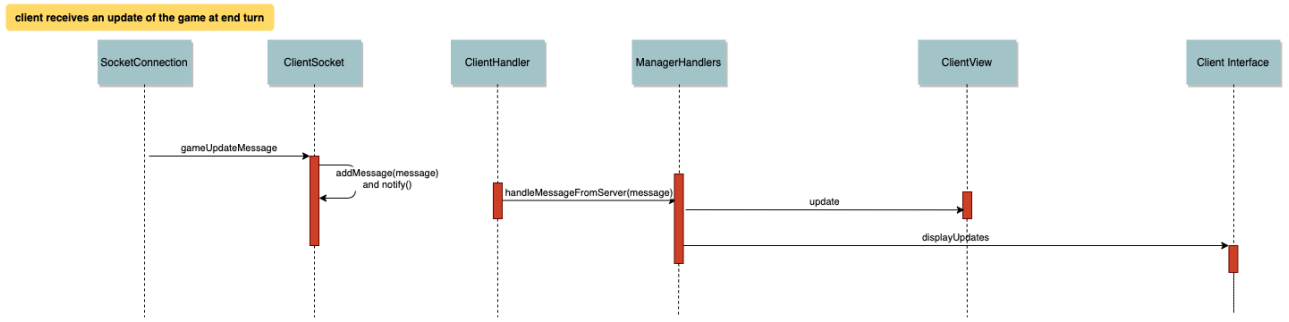
- *KeyAbstractPayload->* TurnPhase.SELECT_COLUMN;
- *VALUE_CLIENT->* column of the bookshelf selected by the user.

c) ACK *(server)*

```
TurnPhase turnPhase=(TurnPhase) newValue;
MessageHeader header=new MessageHeader(MessageType.DATA,playerNickname);
MessagePayload payload=new MessagePayload(turnPhase);
```

The server sends an acknowledgment (ACK) message to the player, indicating the next phase the player should play.

d) END TURN *(server)*




```
MessageHeader header=new MessageHeader(MessageType.DATA,nickname);
MessagePayload payload=new MessagePayload(TurnPhase.END_TURN);
payload.put(Data.NEW_BOARD,boardView);
payload.put(Data.NEXT_PLAYER,modelView.getTurnNickname());
payload.put(Data.TOKEN,modelView.getToken());
payload.put(Data.POINTS,modelView.checkWinner());
payload.put(Data.COMMON_GOAL, modelView.getCommonGoalView());
```

```
payload.put(Data.PERSONAL_POINTS,modelView.getPersonalPoint(nickname));
payload.put(Data.WHO_CHANGE,modelView.findPreviousPlayer().getNickname());
```

At the end of each turn, a customized end turn message is sent to each player, containing individual player-specific data. This includes information that is not accessible to other players, such as the points from personal goals. However, this message is broadcasted to all players to notify them of the turn completion.

It is essential to include the information about the player who has just completed their turn by specifying Data.WHO_CHANGE, to handle disconnections, as the disconnected player's turn would be skipped.

e) ERROR_DATA *(server)*

Types of error data:

```
//Data error
ILLEGAL_TURN("it isn't your turn"),
ILLEGAL_PHASE("This isn't the phase"),
INVALID_COLUMN("Invalid Column"),
INVALID_INPUT("Invalid Input"),
NOT_ENOUGH_FREE_CELLS_COLUMN("You don't have enough cells free in this column"),
NOT_SELECTABLE_TILE("Not selectable tile"),
TOO_MANY_TILES("You cannot select another tile:\n1)you have reached the maximum
number of selectable, \nor 2)bookshelf has a maximum number of free cells lower
than the maximum number."),
INVALID_ORDER_TILE_REPETITION("You cannot enter the same number twice "),
INVALID_ORDER_TILE_NUMBER("You have entered a number outside the permitted range
"),
NOT_VALUE_SELECTED("You have not selected anything"),
INVALID_COORDINATES("COORDINATES of the tile are invalid"),
NOT_SAME_ROW_OR_COLUMN("Tiles are not on the same row or column, tiles must be
all adjacent"),
NOT_ENOUGH_FREE_EDGES("The selected tile has no free edges"),
WRONG_PHASE("You can't do this action now.");
```

Structure error data message *(server):*

```
MessageHeader header=new MessageHeader(MessageType.ERROR,playerNickname);
MessagePayload payload=new MessagePayload(KeyErrorPayload.ERROR_DATA);
payload.put(Data.ERROR,newValue);
```

"newValue" represents the string associated with the error.

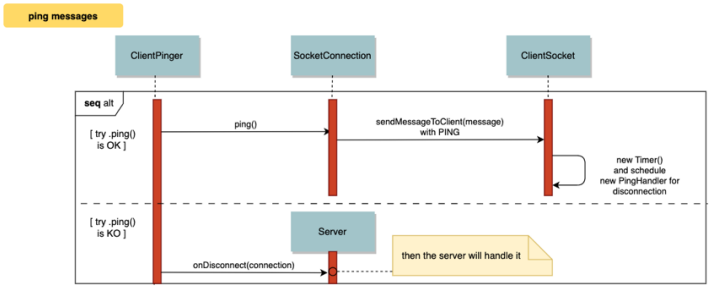
During a player's turn, if the client receives an error data, the player will send a "somethingWrong" message to the server. This message serves as a request for the server to send the "ALL_INFO" message, which contains the necessary game information. Upon receiving the "ALL_INFO" message, the client can reset the attributes of the ClientView and continue the game with updated data.

For data selection, there is a dual verification process in place, both on the client and the server side. This ensures that if the user makes an input error, they do not need to reset the ClientView. This approach also helps to expedite the process, as the error is caught and handled without requiring unnecessary steps or resetting the entire ClientView.

"somethingWrong" message:

- *MessageType*->MessageType.ERROR;
- *KeyAbstractPayload*-> KeyErrorPayload.ERROR_DATA.

4) DISCONNECTION AND RECONNECTION



a) DISCONNECTION *(server)*

```
MessageHeader header = new MessageHeader(MessageType.CONNECTION, nickname);
MessagePayload payload = new MessagePayload(KeyConnectionPayload.BROADCAST);
String content = "Player "+nickname+" disconnected to Game Lobby "+ idGameLobby
+ "!";
payload.put(Data.CONTENT,content);
```

In the event of a disconnection, all other players are notified.

b) RECONNECTION (server)

```
MessageHeader header = new MessageHeader(MessageType.CONNECTION, nickname);
MessagePayload payload=new MessagePayload(KeyConnectionPayload.RECONNECTION);
String content=nickname+" reconnected to Game Lobby "+ idGameLobby + "!";
payload.put(Data.CONTENT,content);
payload.put(Data.WHO_CHANGE,nickname);
```

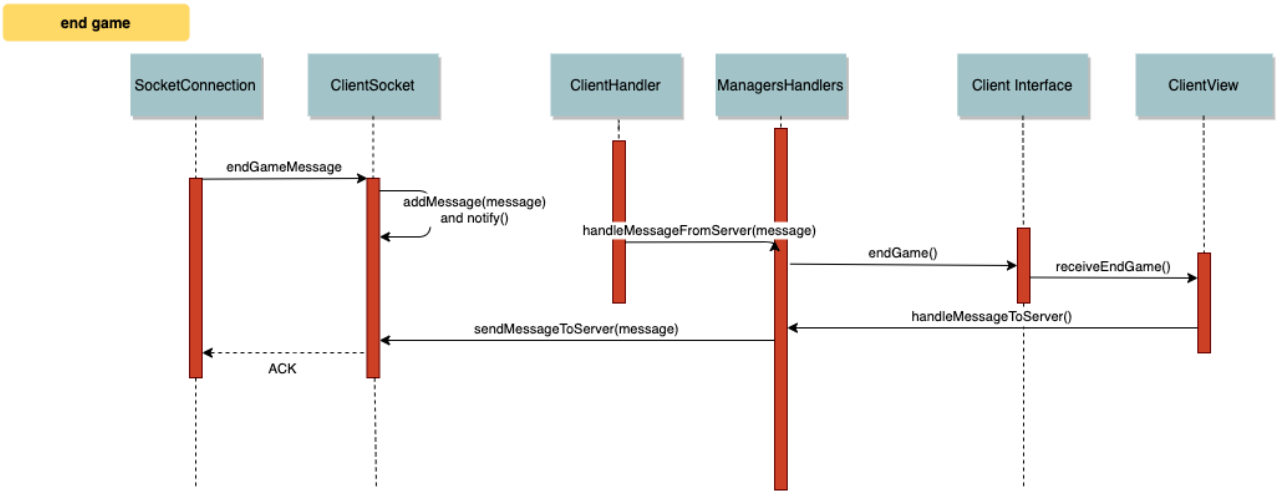
Upon reconnection, the same message is sent to all players, and the player who has reconnected will send a "somethingWrong" message to the server in order to continue the game and reset the ClientView up to that point.

c) ONLY ONE PLAYER IS CONNECTED *(server)*

```
MessageHeader header = new MessageHeader(MessageType.ERROR, nickname);
MessagePayload payload = new MessagePayload(KeyErrorPayload.ERROR_CONNECTION);
payload.put(Data.ERROR,ErrorType.ONLY_PLAYER);
```

If all players, except one, have disconnected, the server will send an "onlyOnePlayer" message. In order to continue the game, at least one additional player needs to connect.

5) END GAME MESSAGE




```
MessageHeader header=new MessageHeader(MessageType.DATA,null);
MessagePayload payload=new MessagePayload(TurnPhase.END_GAME);
payload.put(Data.PERSONAL_POINTS,modelView.getPersonalPoints());
payload.put(Data.POINTS,modelView.getPlayerPoints());
payload.put(Data.BOOKSHELF_FULL_PLAYER,modelView.getBookshelfFullPoints());
Message message=new Message(header,payload);
```

When the game is over, the server sends an "endGame" message to all players. However, the actual termination of the game is deferred until the server receives acknowledgment (ACK) from all players. This approach ensures that even if a player gets disconnected during the game, upon reconnecting with their nickname, they will receive the "endGame" message displaying the final leaderboard. Once the server receives the ACK from the player, it will resend the appropriate "lobby message" related to the decision made in the lobby.

This mechanism guarantees that all players are aware of the game's conclusion. By requiring acknowledgment from all players before finalizing the termination, it ensures that no player misses out on the endGame message, regardless of their connectivity status during the game.

6) OTHER ERRORS (CONNECTION AND LOBBY)

```
//Connection error
PING_NOT_RECEIVED("Ping not received. Disconnection started.."),
DISCONNECTION_FORCED("Something went wrong during login. Connection refused!
Disconnection started.."),
DISCONNECTION("Disconnection from the server"),
ERROR_CONNECTION("Error in creating connection, try again"),

ERR_NICKNAME_LENGTH("Invalid username length. Disconnection.."),
ERR_NICKNAME_TAKEN("This nickname is already taken  Disconnection.."),
ONLY_PLAYER("You are the only player connected, wait for another to continue the
game"),


//Lobby error
ERR_NUM_PLAYER_WANTED("You must select a number of players between 2 and 4.  You
```

```
will be sent back to GlobalLobby.."),
ERR_JOINING_GAME_LOBBY("Failed in joining the Game Lobby! You will be sent back
to GlobalLobby.."),
ERR_GAME_FULL("Failed in joining the requested game lobby because is full! You
will be sent back to GlobalLobby.."),
ERR_GAME_NOT_FOUND("Failed in joining the requested game lobby because it
doesn't exist! You will be sent back to GlobalLobby.."),

ERR_NO_FREE_SPOTS("Failed in joining a random game lobby because all games are
full!Creating a new Game Lobby for min num players..."),

ERR_RECONNECT_TO_GAME_LOBBY("Failed to reconnect to previous game lobby!
Disconnection.."),
ERR_JOIN_GLOBAL_LOBBY("Failed in joining the Global Lobby! Disconnection..");
```