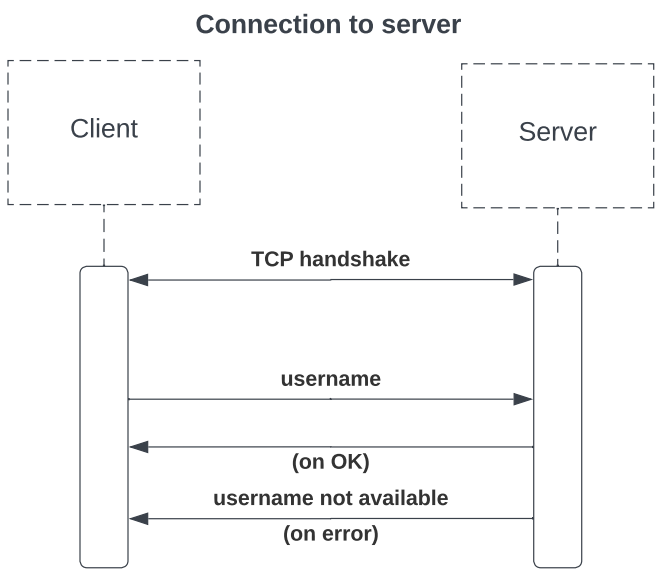
Given the architecture of our model (changes are applied via delta java objects), we are plannning to send all data using serialized java objects, by making use of the Serializable java interface. We'll be using Message objects with a header field describing the logical type (e.g: Control, GameAction, Error, etc...) followed by the relevant data to that type.

Server and clients remain aware of each other using ping-pong messages that start from the server:

- If a client doesn't respond to a fixed amount of pings from the server the server assumes he lost connection
- if a client doesn't receive pings for a fixed amount of time the client assumes the server crashed

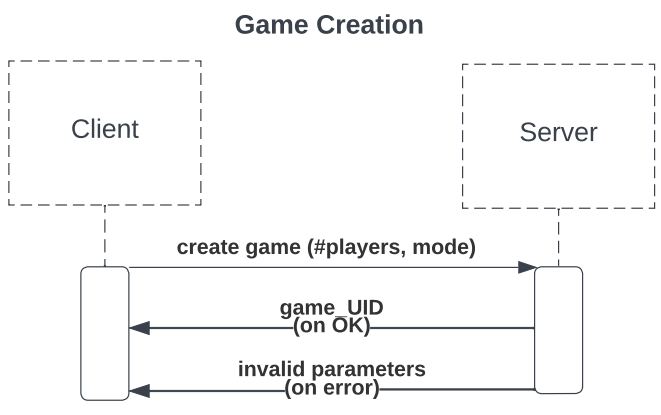
Connection to server:

- TCP handshake between client and server
- client sends a message with his username
- server validates username
  - if its valid: client can now Create Game/ Join Game
  - else: client has to send a new username



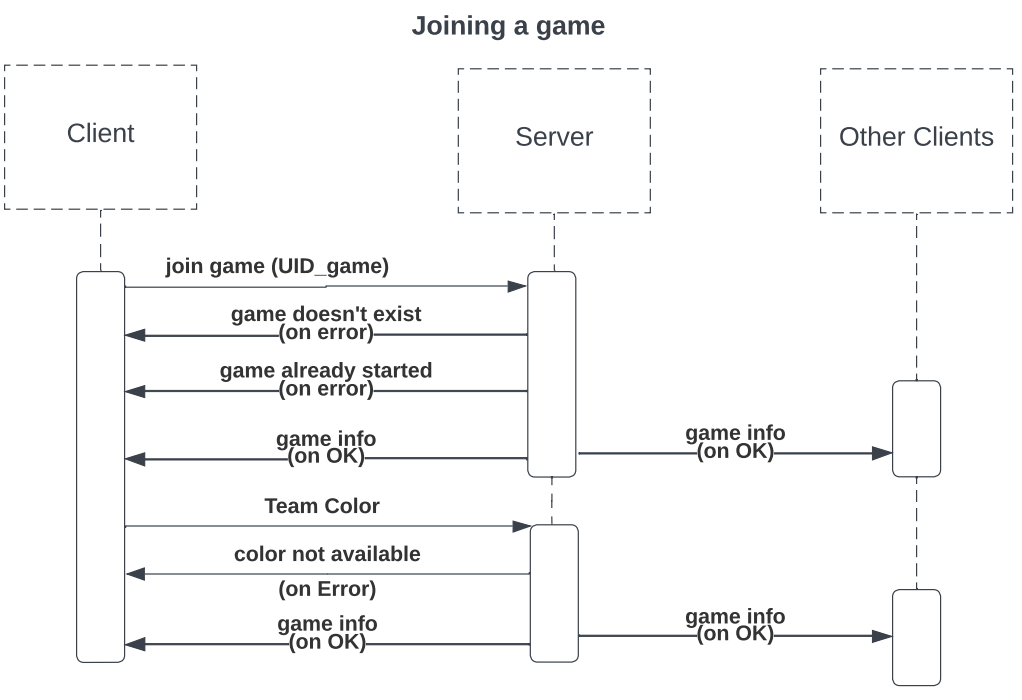
Game creation:

- client creates a game providing (#players, mode)
- server checks validity
  - if the parameters are valid: server sends back game_UID (unique ID) , client goes in waiting room
  - if the parameters are invalid, the server sends back an error



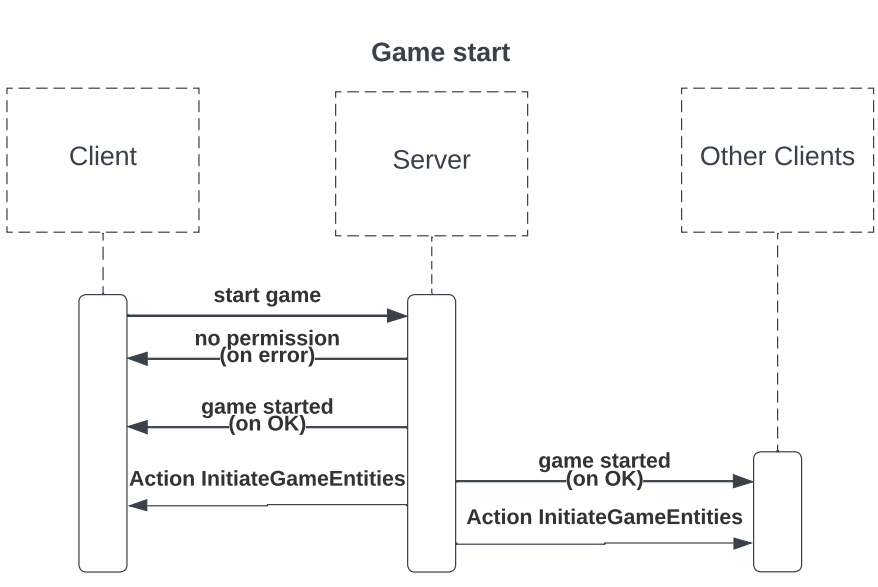
Joining a game:

- client joins a game providing a game_UID
- server looks for a game that has the provided game_UID
  - if the game exists and is valid sends updated **game info:** (#player , game_mode, nicknames + Teamcolor) to all clients
  - if the game doesn't exist, the server sends back an error
  - if the game is already in progress (and the player isn't rejoining), the server sends back an error
- client sends a Team color
- server checks color validity
- server resends **game info** to each client



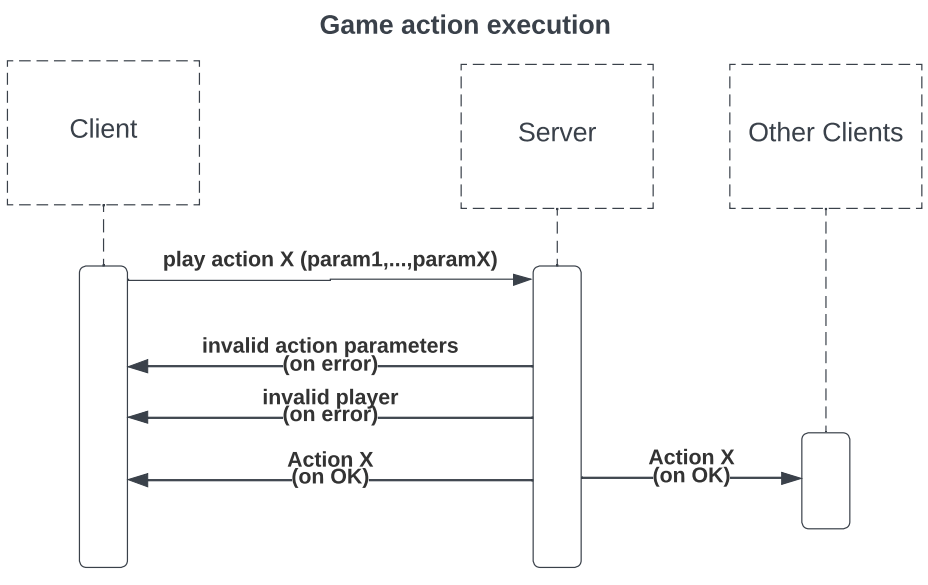
Game start:

- client starts game
- server checks client validity (client == Owner?):
  - if valid:
    - server sends to all clients a game started message
    - server initializes gameState and sends InitiateGameEntities Action to all clients
  - else: no permission error



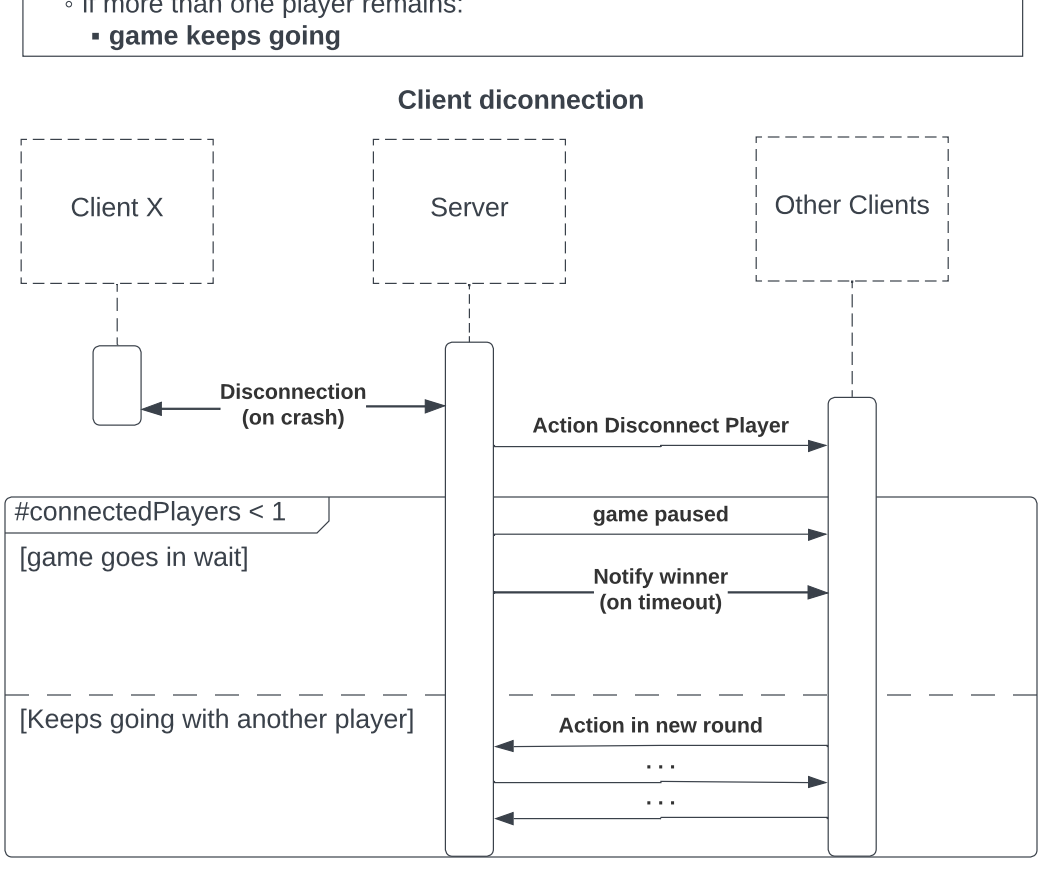
Game action execution:

- client comunicates the action he wants to play
- server checks validity:
  - if valid, the server changes its gameState with the corresponding Action and forwards it to all clients
  - else: sends back error messages



Client disconnection:

- a client loses connection
- server:
  - after some unrequited pings, it changes the player state with the DisconnectPlayer action and forwards it to all clients
  - less than two players remain:
    - game pauses starting a timeout
    - sends a game paused message to the remaining client
  - if more than one player remains:
    - **game keeps going**



Client reconnects to a game:

- clients joins a game providing a game_UID
- il server checks if he is a disconnected player
  - if he is: the server updates the player state with a ReconnectPlayer Action and then forwards it to all clients
- if he isnt: error game already in progress