AM15
Caspani-Citterio-Barbiero

COMMUNICATION PROTOCOL

0. INTRODUCTION

The game communication protocol is based on the exchange of Message objects, serialized in human readable JSON format.
There are two typologies of message: **Client-to-Server** and **Server-to-Client**.
The client can always receive update notifications from the server but he can send messages only in certain moments of the game.
If a message is sent in an inappropriate moment, this will be discarded and not processed by the server.
During his turn, the client receives messages from the server that notify him about the progress of his moves, in order to achieve a cleaner View behaviour.

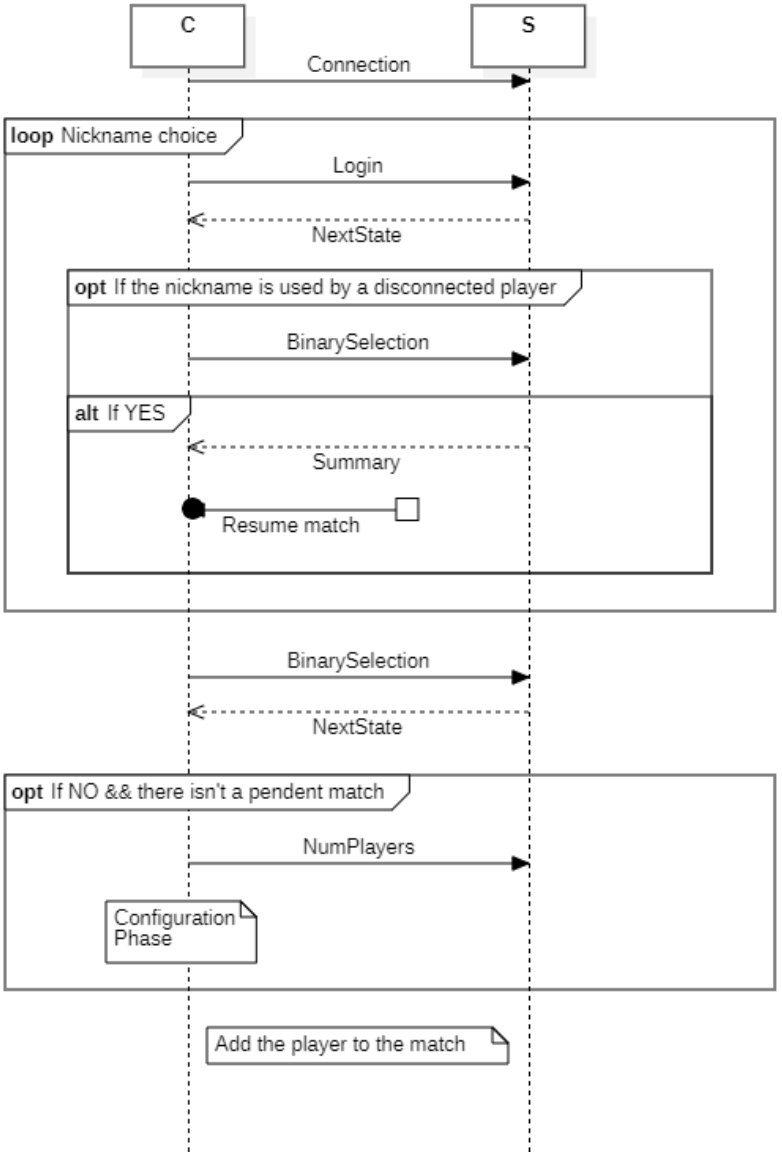
From now on the client entity will be called “C”, whereas the server will be “S”.

1. CONNECTION AND RECONNECTION PHASE

When C connects to S, he has to send a LoginMessage with the chosen nickname.
S will answer him with a NextStateMessage.

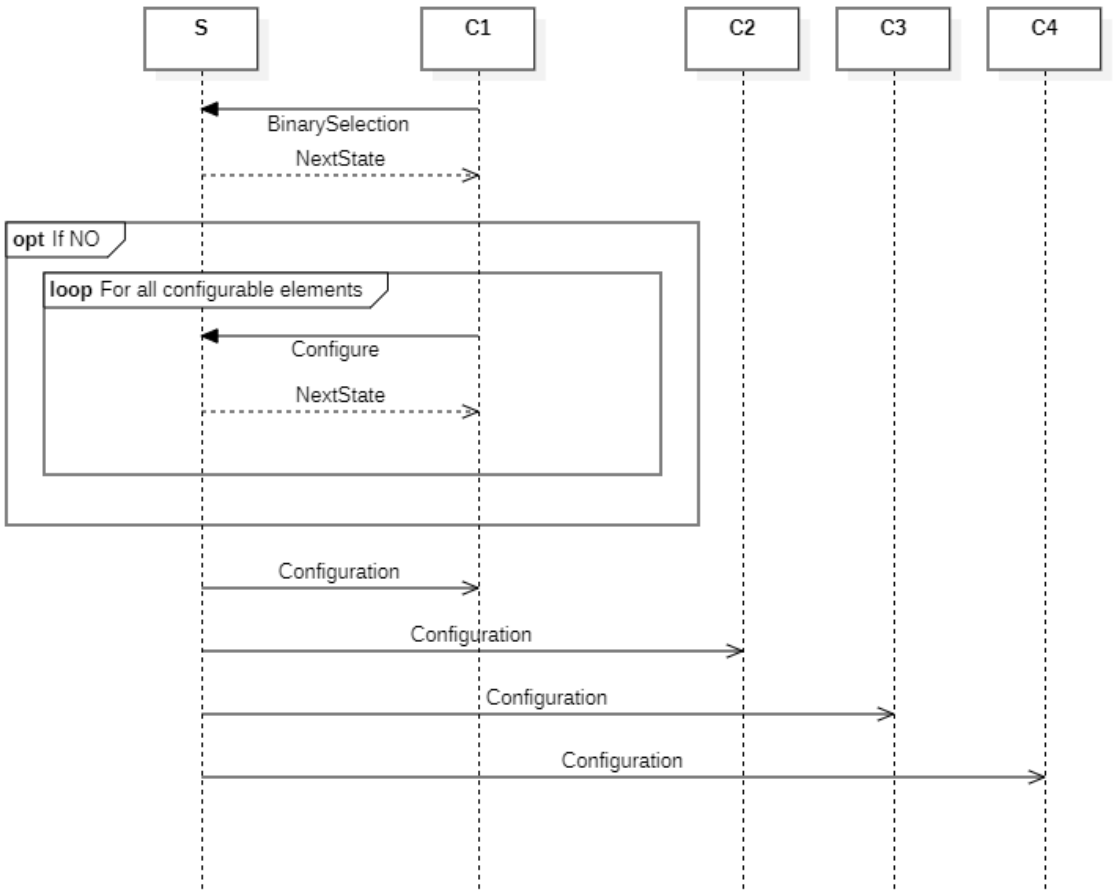
If the nickname is linked to a previously disconnected client, C is supposed to choose if he wants to continue the current match via a BynarySelectionMessage. In case of a positive answer S will send a SummaryMessage to C and C will resume from where he left, otherwise S will wait for another LoginMessage.

If the nickname is in use among the active players connected to the server, the client will be asked to choose a new one via a RetryMessage;
Next, C selects through a BinarySelectionMessage if he wants to play a singleplayer match.
In the case of multiplayer choice, if there isn't another forming match, C has to select how many players he wants in his match through a NumPlayersMessage, otherwise he will be added to the forming one.



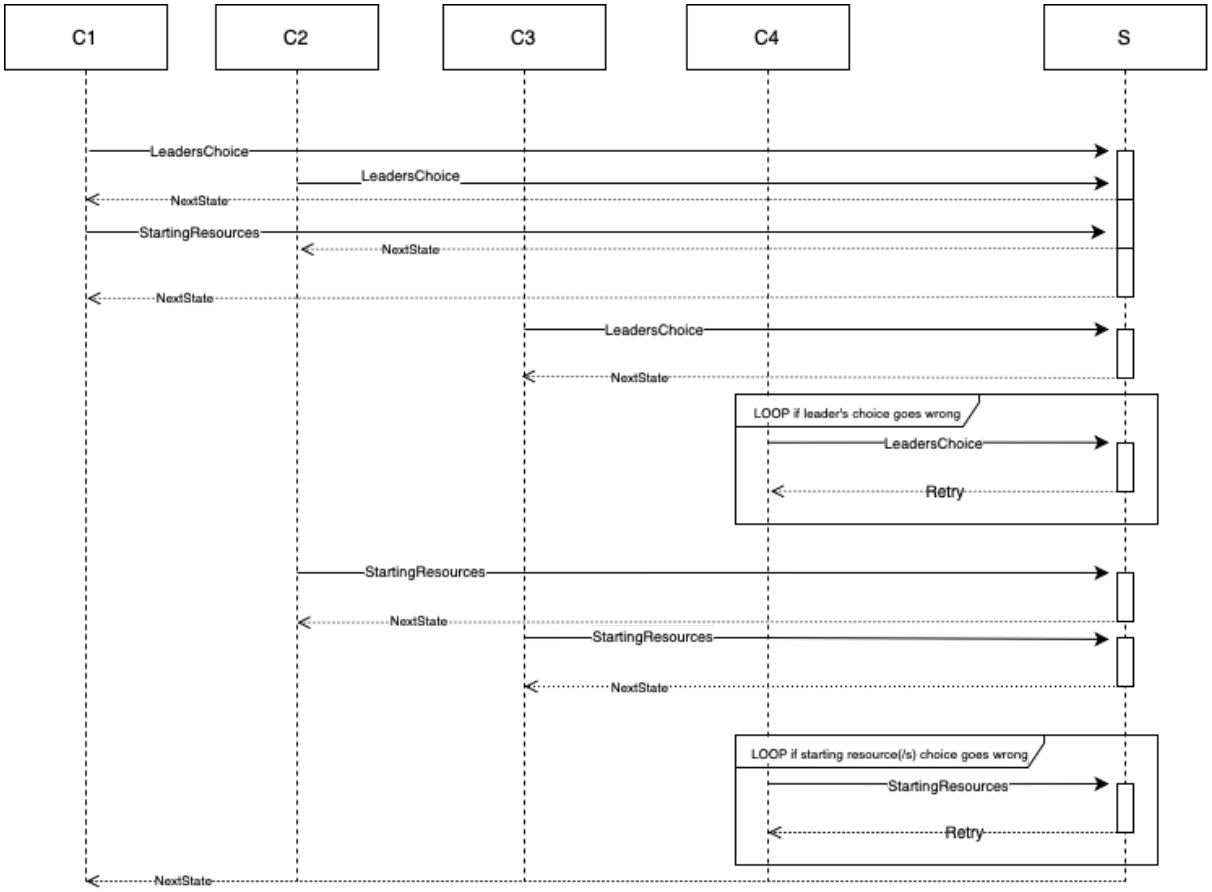
2. *CONFIGURATION PHASE*

In the configuration phase the clients have to set up the game and so they need several pieces of information. These are sended to them by S after that the client who created the match has sent to S the chosen configuration. First C sends a BinarySelectionMessage to choose between default or custom configuration, in case of custom configuration C and S exchange messages to validate pieces of configuration.



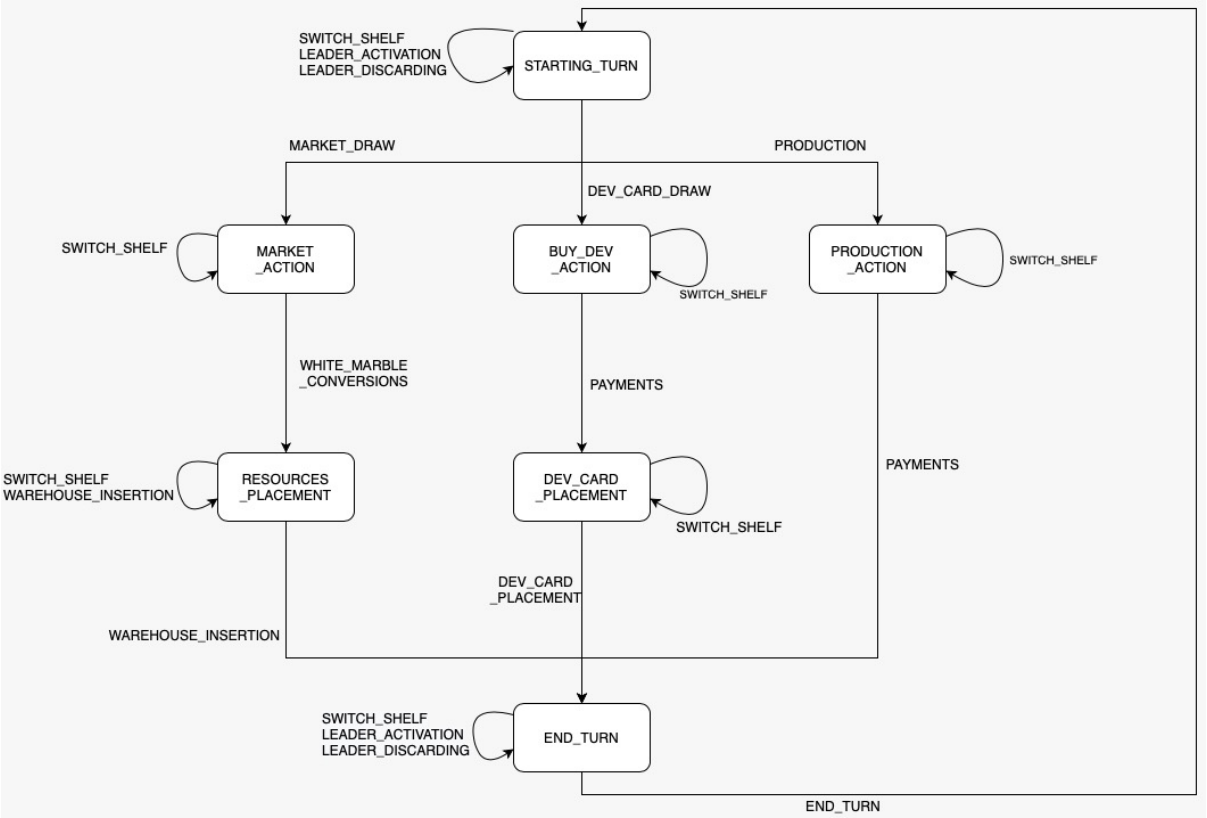
3. *STARTING GAME PHASE*

In the starting game phase every player in the match, depending on his position on the cycle, chooses his leaders and starting resources simultaneously.
Every player's choice will be sequentially taken and processed, associating a State for everyone in the match.
When every player is in the STARTING_PHASE_DONE state, the match starts.



> LeadersChoiceMessage:
[String nickname, List<String> CardIDs]
> StartingResourcesMessage:
[String nickname, List<PhysicalResource>]
> RetryMessage:
[String nickname, String errMessage]
> NextStateMessage:
[String nickname, StateName newState]

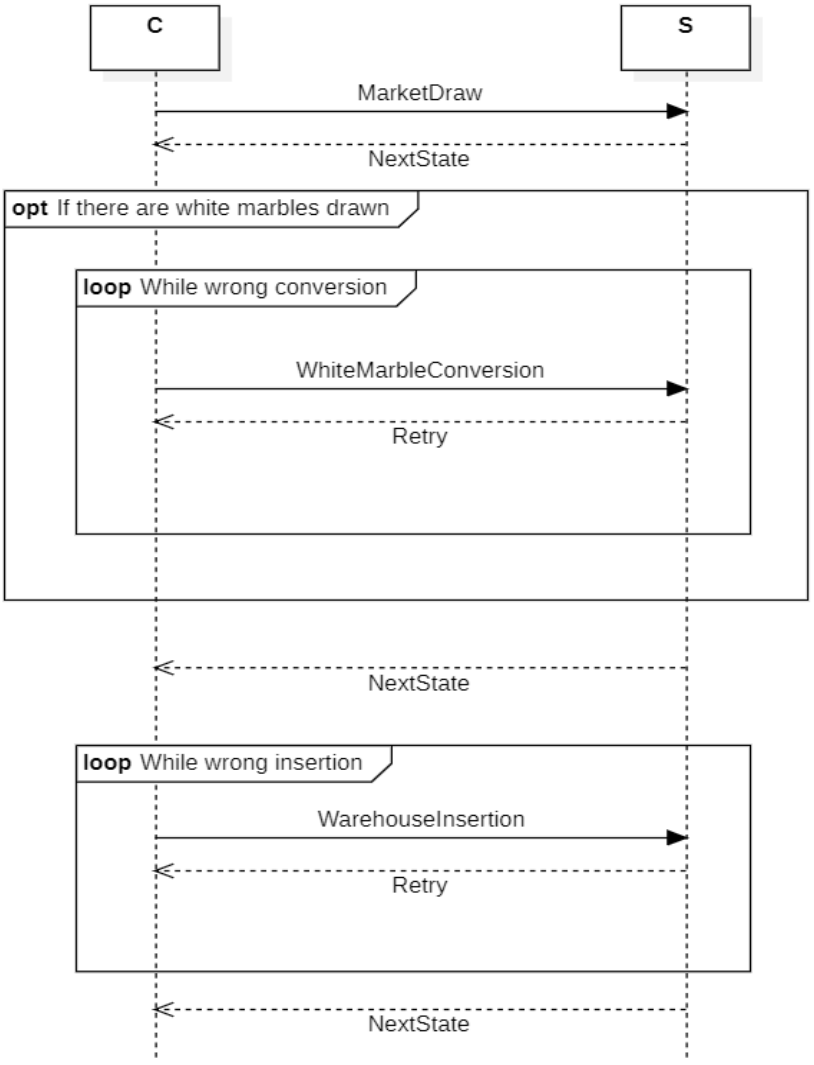
4. TURN PHASE



The turn phase handles all the main actions of the game by a turn-based logic, indeed the only player who can send command messages is the current player and, according to the fsm's state, only some messages are accepted.
If a message comes from another player it will be ignored, else if the message comes from the right player but in the wrong fsm state the player will receive a RetryMessage from S.
The turn phase always starts in the STARTING_TURN state and evolves according to the received messages.
LeaderActivationMessage and LeaderDiscardingMessage are accepted only in STARTING_TURN and in END_TURN, SwitchShelfMessages are always accepted and they don't change the fsm's state.
MarketDrawMessages, DevCardMessages and ProductionMessages evolve the fsm into three possible branches: market branch, dev card draw branch and production branch.

4.1 MARKET DRAW PHASE

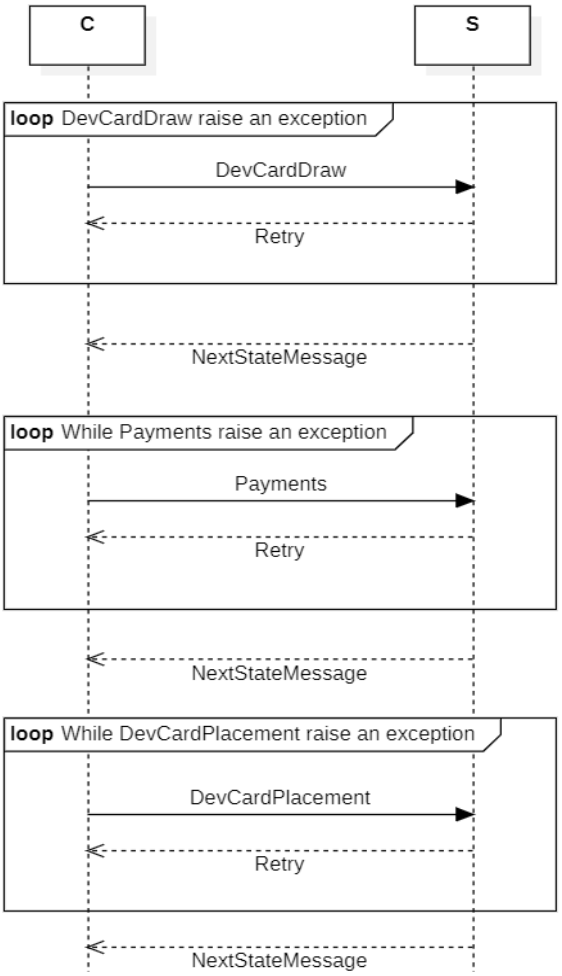
When C sand a MarketDrawMessage the fsm enters in the markert draw phase.
If the message's parameters are wrong S will sand a RetryMessage to C, else
the number of white marbles drawn will define the next state in the fsm.
If that number is more than 0, the controller changes state to MARKET_ACTION and S will expect a WhiteMarblesConversionMessage and after that a WarehouseInsertionMessage.
If that number is 0, the controller changes state to RESOURCES_PLACEMENT and S will expect a WarehouseInsertionMessage.
When it comes, S will send a RetryMessage to C if the message contains an invalid move, else the controller changes state to END_TURN.



> MarketDrawMessage:
[String nickname, boolean row, int num]
> WhiteMarblesConversionMessage:
[String nickname, List<PhysicalResource>]
> WarehouseInsertionMessage:
[String nickname, List<PhysicalResource>]

4.2 DEV CARD DRAW PHASE

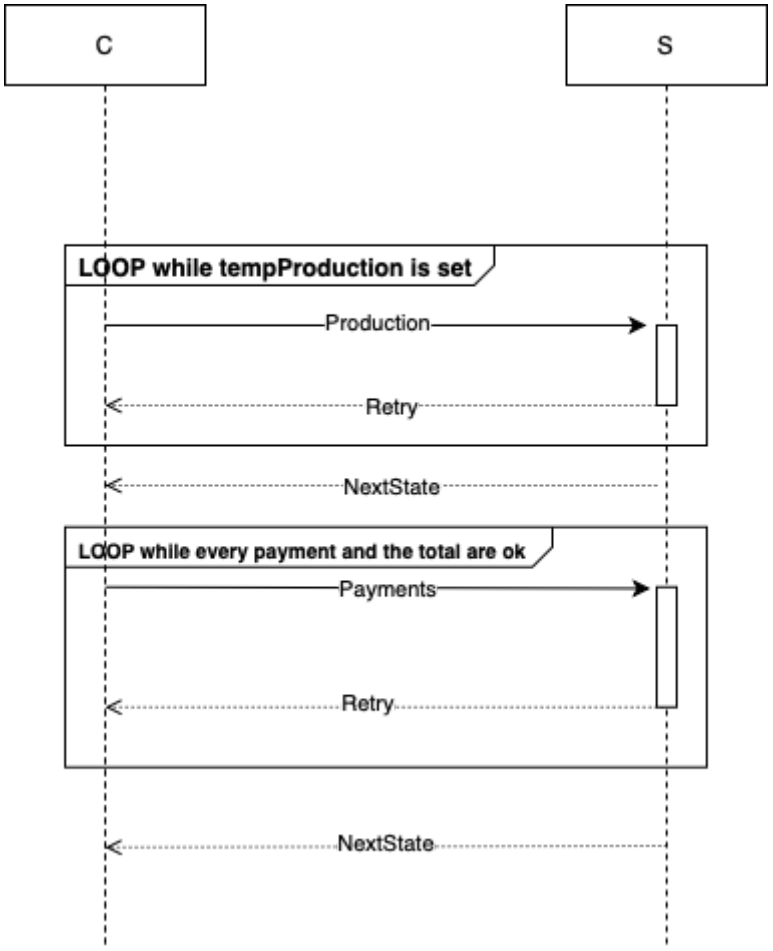
When C sends a DevCardDrawMessage the fsm enters in the dev card draw phase.
If the card selected can't be taken by C then S will send a RetryMessage to C, else the controller changes state to BUY_DEV_ACTION and waits for a PaymentsMessage.
When the PaymentsMessage comes, if its parameters are wrong, S will send a RetryMessage to C.
When the PaymentsMessage is accepted, the controller changes state to DEV_CARD_PLACEMENT and waits for a DevCardPlacementMessage.
When it comes, if its parameters are wrong, S will send a RetryMessage to C, else the controller changes state to END_TURN.



> DevCardDrawMessage:
[String nickname, int row, int column]
> PaymentsMessage:
[String nickname, List<PhysicalResource>, Map<Integer, PhysicalResource>]
> DevCardPlacementMessage:
[String nickname, int column]

4.3 PRODUCTION PHASE

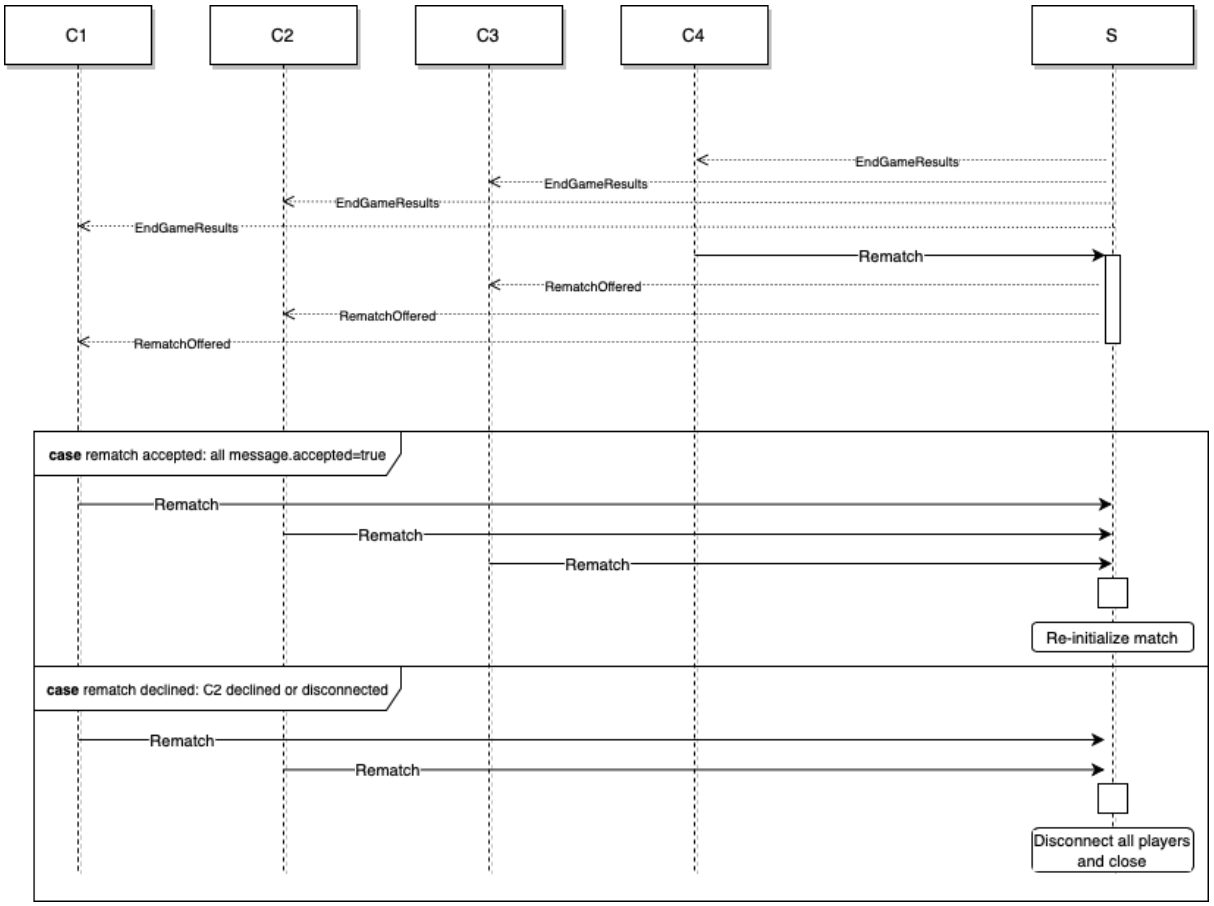
When C chooses to perform a production action, he's supposed to send to S a specific message that contains the list of all the chosen producible cards'IDs and a single Production which specifies the whole conversion of costs and earnings of Unknown resources.
Error (Retry) messages are thrown in case of: not-producible card choosing, not available resources as payments, mismatch between unknown resources and chosen cards.



> ProductionMessage:
[String nickname, List<String> CardIDs, Production unknownProduction]
> PaymentsMessage:
[String nickname, List<PhysicalResource> sbPayments, Map<Integer, PhysicalResource> whPayments]
> RetryMessage:
[String nickname, String errMessage]
> NextStateMessage:
[String nickname, StateName newState]

5. END OF MATCH PHASE

When the last turn has been played by everyone, the game evolves again into a state in which S broadcasts the ranking of the match, one player can offer a rematch and the others can accept or decline it simultaneously.
If no one declines/disconnects then the game restarts with the same configuration, else the match will be completely closed and the players “gently” disconnected.



> EndGameResultsMessage:
[String nickname, Map<String,Integer> leaderBoard]
> RematchMessage:
[String nickname, boolean accepted]
> RematchOfferedMessage:
[String nickname, String message]

6. DISCONNECTION

In order to achieve a fast detection of the disconnections, C is supposed to send a PingMessage periodically to S every few seconds. The same from S to C.
If C doesn't send a message until the timeout expires, the player is set disconnected.
If S doesn't send a message until the timeout expires, the player can consider the server as disconnected.