Gruppo A35 – Donati-Di Marco-Dell'Agosto

CLIENT SERVER COMMUNICATION PROTOCOL

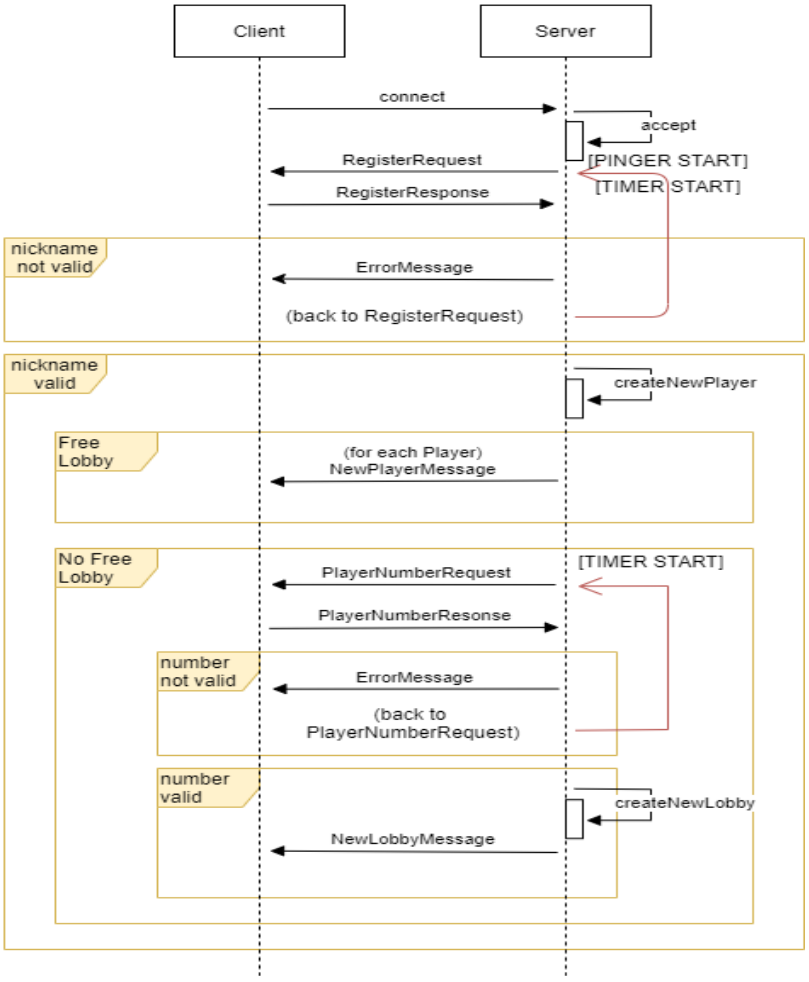
We decided to represent the communication protocol through 6 different phases. These do not directly describe the phases that divide the game, but all those that require a precise specification at the Client-Server communication level.

The connection of several players was managed through a queue for access to a lobby, which is created if the newly connected one is the first player or the previous lobbies are full. Obviously, if there are free lobbies, they are filled with the current players in the queue. Each player attempting a connection must first provide a valid nickname. The choice of the Nickname is important because in case of disconnection the progress of that player remains precisely assigned to that specific player.

The whole game is divided into 3 different phases, which are dealt with in detail later:

- SETUP: first connection of the client, with choice of nickname and assignment to a Lobby. Choice of Leader Cards and assignment of any bonus resources for each player;
- ONGOING: This phase is the longest, each player during his turn can choose one of the three different main actions: Buying resources from the market, Buying a Development Card or Activating a Production;
- ENDGAME: Once one of the two End Game conditions has been reached (Achievement of 24 Faith points or the Purchase of the 7th Development Card for a player), the scores of each player are calculated and the winner is announced

FIRST CONNECTION

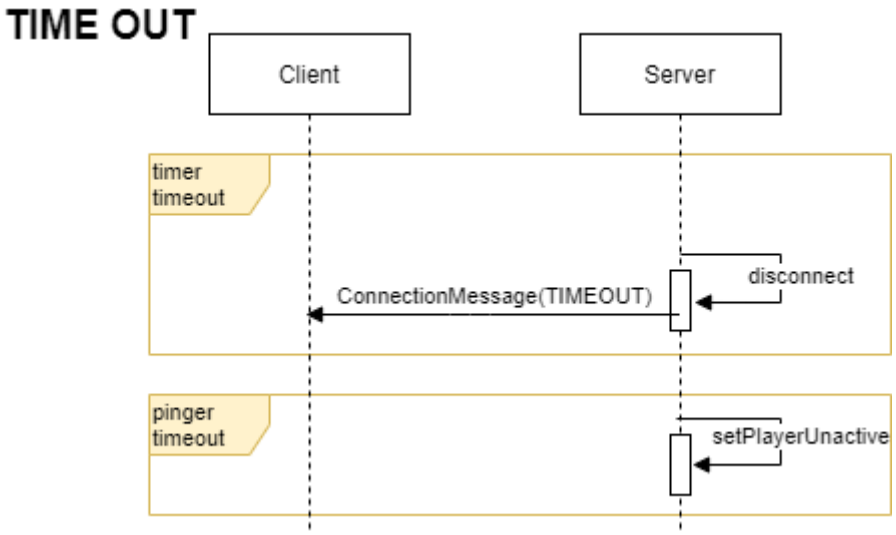


SETUP

The player tries to connect to the Server and immediately the Server respondes asking the Player's Nickname. As soon as the server receives the Client's request a pinger will start and its duty is to check if the player pinged is still connected to the Server. When the Server asks for the Nickname, puts itself in wait for the Client's response and creates a timer and a counter. This timer is used to check how long the response took and how many times the Player tries to insert a valid nickname. If the timer expires or the counter reaches a high number? the current player is automatically disconnected.

Once the player's nickname has been sent, the Server checks its validity. An invalid Nickname matches to an Error Message and player is invited to re-do the Register Request. Otherwise If the sent nickname is valid, a Virtual Client is created and the current player added to the queue.

Now the Server checks for a free Lobby. If a free Lobby is found, the current player is added to that, differently, due to the fact that all lobbies are full, the Server asks the number of players for that match. The Server checks the provided number and, eventually, an Error Message is sent back to the Client bringing it back to the Player Number Request, asking to provide a valid number. If the number is valid, a new Lobby is created and the current player notified of the Lobby creation.



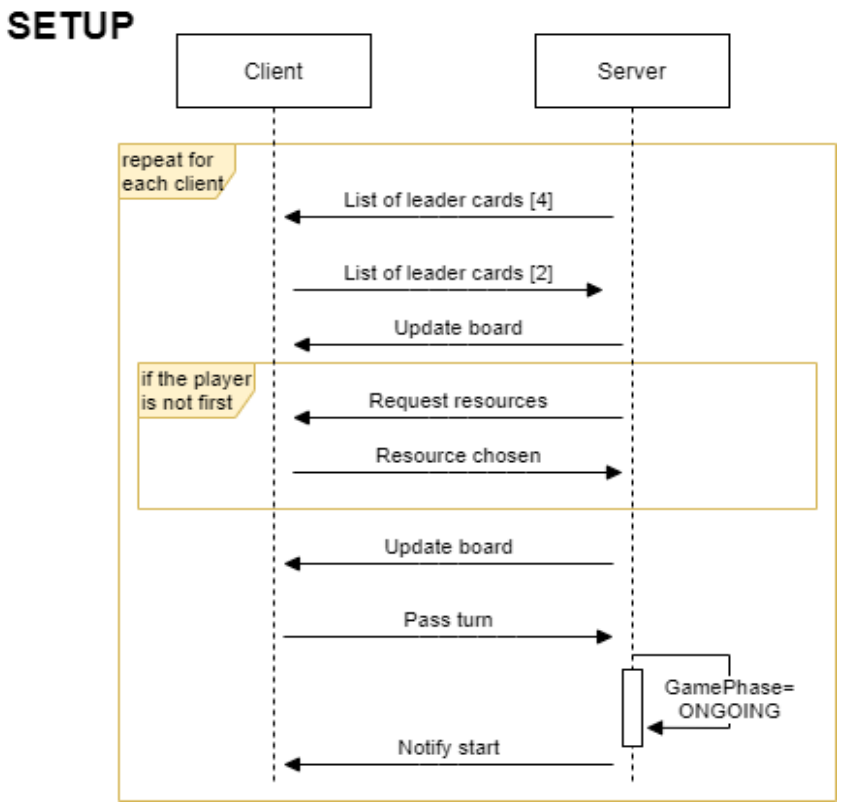
The above sequence diagram shows how the Server reacts to a too long waiting for a Client's response. Whether the Player takes too much to send the Nickname or the Number of Players.

DISCONNECTION

A player's disconnections managed as follows. The player's nickname has to be unique exactly for this case. When he tries to reconnect, entering his own name he can get back in the game without losing anything, due to the fact that the name is unique and corresponds to one and only one player in game.

We can discerne between two cases:

- The game phase is already ongoing, the server just notify all the players;
- The Player is still in the Lobby and the game isn't started yet, the player is removed from the lobby and the other players notified.



PLAYER SETUP

Following a random player Order sequence chosen as soon as the game starts, for each player in the Lobby, the Server sends a list of 4 Leader Cards and the respective player has to choose 2 of them. As soon as the server gets the player response the Board is Updated and the Server proceedes to distribute the bonus resources from the second player on.

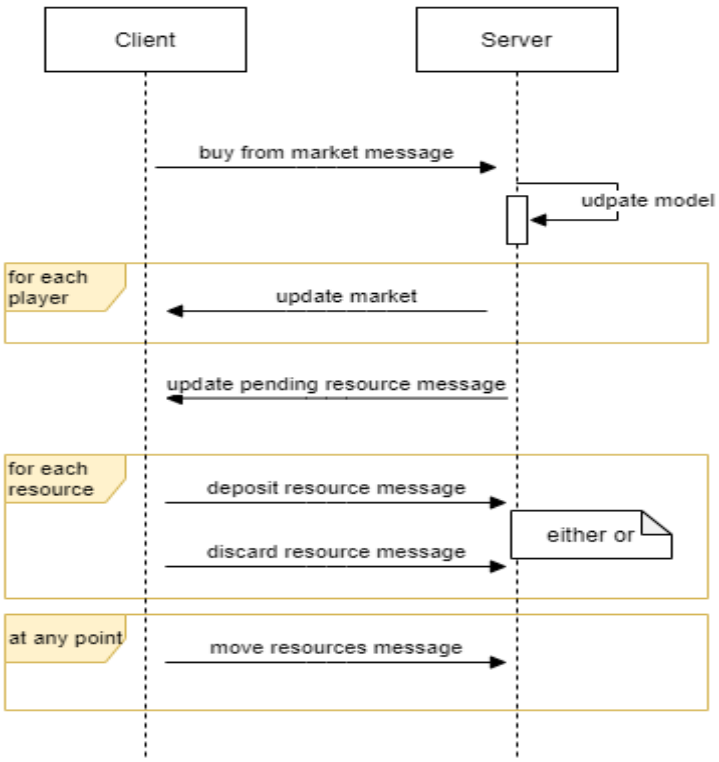
This concludes the Setup Phase, from this moment on the game enters its main phase.

GAME START

Now, in this game phase the Server checks if the player is connected. If it's not the current turn is skipped and the next player's turn begins. Then the server waits for the Player to choose among one of the main action. If the chosen action is not valid the Server responds with an Error Message and invites the Player to do an other action, this time hoping it's legal. Otherwise the Server resolves the request and updates the whole board. With the update the Server has to know if the turn is ended or not. Assuming that the turn is not ended and the player is allowed to do another action, the interaction Client-Server is the same as the previous one, else Server has also to check if it isn't also the last game turn (The game has entered in the ENDGAME phase). In the positive case the next player's turn has begun and the current player notified, otherwise the game has ended, all the player notified, results calculated and winner announced.

As written before, each player's turn is based on various minor action or one main action between these three:

BUY FROM MARKET

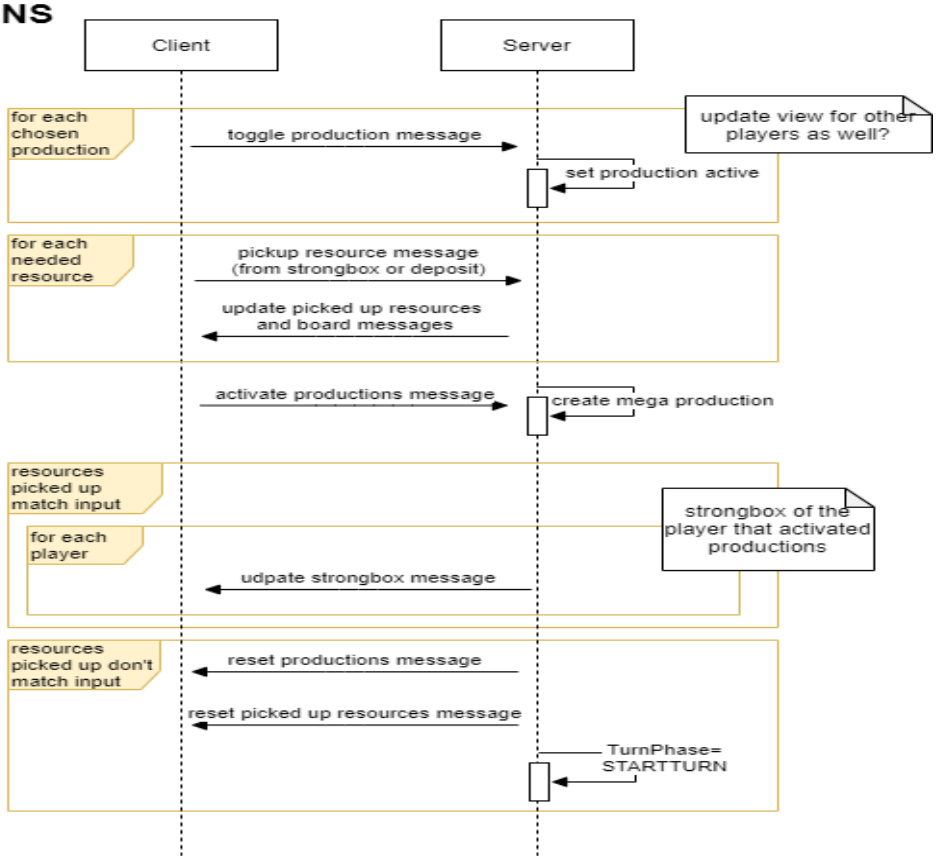


When a Player decides to acquire resources from the Market, it has to notify the Server that has to update the current and the other players' model. The Server also updates the pending resource with a relative message, due to the fact that can't be pending resources if the player's wants to pass the turn.

For each resource the current palyer has in the pending list can send one between two messages: it can decide to deposit a resource in pending or either discard it.

Also, the depositing of a resource can be preceded by a move resource message due to the fact that the Main Depot has strict rules in the resource depositing.

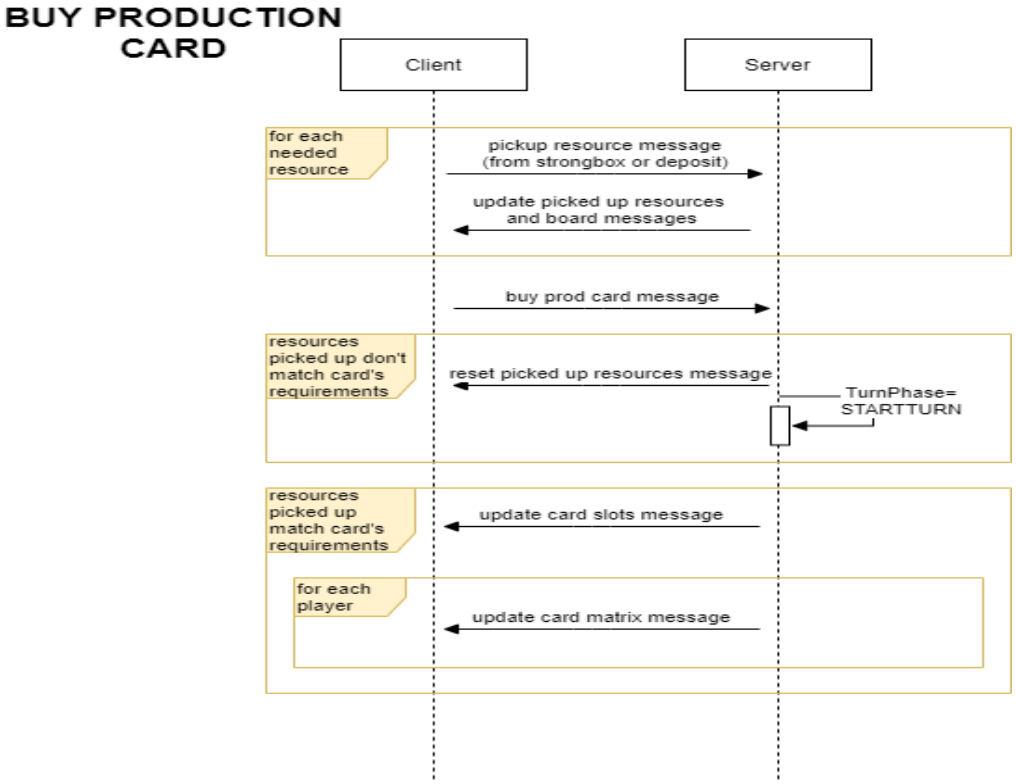
PRODUCTIONS



If the player decides that his main actions consists in a Production, he can choose among all the avaiable production, from the Base Production, always avaiable, to the Development Cards production.

For each chosen production the Client has to send a message containing all the toggled production, the Server sets the relative production active. Each production has its input and output, for this reason the Client has to send all the resource needed for the toggled production in a message while the Server updates the storing system of that player. Then the Player has to communicate the final message that consists in the activation of all the toggled productions.

Obviously there could be problems. If the resources matches the production's input the only task of the Server is to update the player's strongbox and the whole game board. Else the Server resets every production and sends a message, after the Server resets all the previously picked resources and reverts the game phase to the START TURN



The last main action is buying a Development Card. The Player has to send a message containing all the needed resources present in the card the player wants to buy, the Server responce is simply an update of the board regarding the picked resources.

Now as soon as the Player's ready can comunicate which card he wants to buy. As the previous action there could be 2 possible actions: The picked resource don't match the requirements, so the Server has to revert the picked up resources and set the phase back to START TURN; Or, if matches, the server has to update the player's card slot with the just bought card and has to update the global card matrix for each player in Lobby.

MESSAGES

We decided to use json for all the messages delivered between client and server.

SETUP PHASE

RegisterRequest: {“message”:”insert nickname”}

RegisterResponse: {“nickname”:value}

GenericMessage: {“message”:”text”} (es. connection/disconnection notify, new lobby message,…)

ErrorMessage: {“error”:”text”}

PlayerNumberRequest: {“message”:”insert number of players”}

PlayerNumberResponse: {“number”:value}

ConnectionMessage(TIMEOUT) : {“message”:”waited too long” }

USER ACTIONS

Those are the commands sent from the user to the server referred to the public interface of the model:

public void chooseLeader(List<LeaderCard> l)

command:

```
[
 {"id":"nome_carta1.png"},
 {"id":"nome_carta2.png"}
]
```

---

public void chooseResourceToDeposit(Integer id,ResourceType res)
public void depositResource(Integer id,ResourceType res)

command:

{"id":intero, "res":resourceType}

---

public void playLeader(int index)
public void discardLeader(int index)

command:

{"id":"nome_carta1.png"}

---

public void buyAtMarketInterface(char rc,int index)

command:

{"rc": ‘char’, "index": int }

---

public void discardResource(ResourceType res)
public void transformWhiteIn(ResourceType res)
public void pickUpResourceFromStrongbox(ResourceType res)
public void toggleDiscount(ResourceType res)

command:

{"res": “resourceType”}

---

public void substituteUnknownInInputBaseProduction(ResourceType res)
public void substituteUnknownInOutputBaseProduction(ResourceType res)

command:

{"res": “resourceType”}

---

public void substituteUnknownInInputExtraProduction(Integer index,ResourceType res)
public void substituteUnknownInOutputExtraProduction(Integer index,ResourceType res)

command:

{"id":intero, "res":”resourceType”}

---

public void toggleExtraProd(Integer index)

command:

{"id":intero}

---

public void toggleCardProd(Integer slot)

command:

{"slot":intero}

---

public void pickUpResourceFromWarehouse(Integer id)

command:

{"id":intero}

---

public void buyCard(Integer row,Integer col,Integer slot)

command:

{"row":intero, "column":intero, "slot":intero}

---

public void moveResource(Integer dep1,Integer dep2)

command:

{"id1":intero, "id2":intero}

---

public void toggleBaseProd()
public void activateProductions()
public void revertPickUp()
public void passTurn()

command: {}