

# Communication Protocol

For the communication messages between the client and the server, we decided to use JSON format in support of an *event-driven* implementation.

Some similar types of messages are used in different phases, so, in order to distinguish them, we defined the following common JSON scripts, characterized by different types:

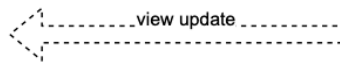
## FROM SERVER TO CLIENT:

### 1. string message:

```
{
  "type": "error/info"
  "payload": "String containing the message"
}
```

### 2. view update message:

This kind of messages are sent to every client to update the gameboard's state and/or the personal board of some players. They are represented in the following UML sequence diagrams with this arrow:



```
{
  "type": "graphicUpdate",
  "marketUpdate": ["red", "yellow", "white", "green", "blue", ... , yellow"], (first: extra slot)
  "gridUpdate": {
    "fullGrid": ["id1", "id2", ... , id12"] (full visible grid for initial/reconnection update)
    "level": 2,
    "color": "yellow",
    "newCard": "id" ("empty" if empty slot)
  },
  "personalBoardUpdateList": [
    {.(following attributes are present only when needed)
      "nickname": "nickname",
      "handLeaders": ["id1, id2"],
      "activeLeaders": ["id1, id2"],
      "productionBoard": ["id1, id2, id3"],
      "warehouse": [
        {5 : "blue"}, {20 : "EMPTY_RES" ("EMPTY_RES" if that slot is now empty)}
        ... (for all resources)
      ],
    },
  ],
  "faithTrackUpdate": {
    "indexes": {"nick1": 8, "nick2": 2},
    "reports": {
      "nick1": [boolean, false, false],
      "nick2": [false, false, false]
    }
  }
}
```

### 3. choice message:

This kind of messages are used when the player has to make a decision (requested by the server) after an *action*

```
{
  "type": "choice", (different for every choiceAction)
  "nickname": "nickname"

  "numberTransformation": 2,
  "possibleTransformation": ["blue", "green"],
or
  "newCardID": "id"
}
```

## FROM CLIENT TO SERVER:

### 4. action:

All the possible *actions* a player can do are sent with this format:

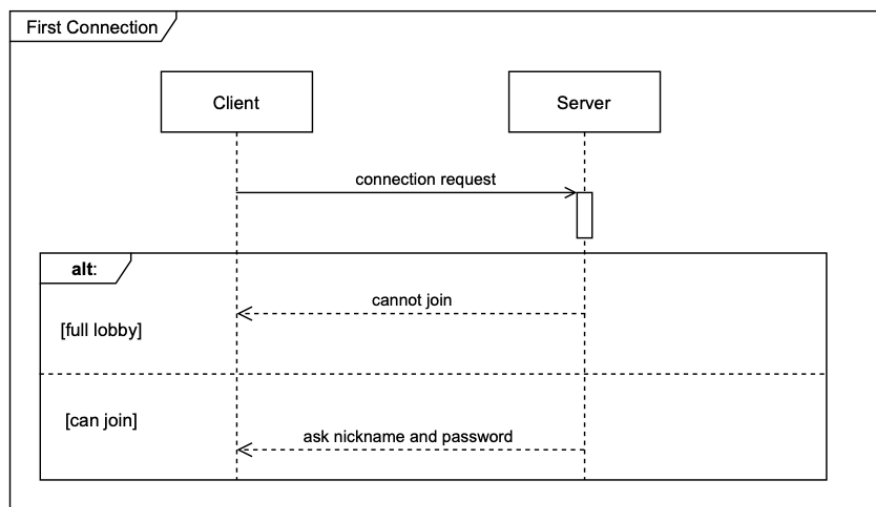
```
{
  "sender": "nickname",
  "type": "leaderAction, buyAction ... ",

  "param1": "id",
  ...
  "paramN": true
}
```

## Client-Server communication

Convention: in all phases, the server will ask the same request to the client until the answer is valid.

### 1 - Connection Phase



The *connection phase* starts with a client connecting to the server through a socket. If the lobby is not full, the server immediately sends a **string message** requesting to insert a (valid) nickname and password; else the server send a **string message** informing that connection has been denied.

### 2 - Login Phase

- TO SERVER: nickname and password

```
{
  "type": "login"
  "nickname": "nick"
  "password": "psw"
}
```

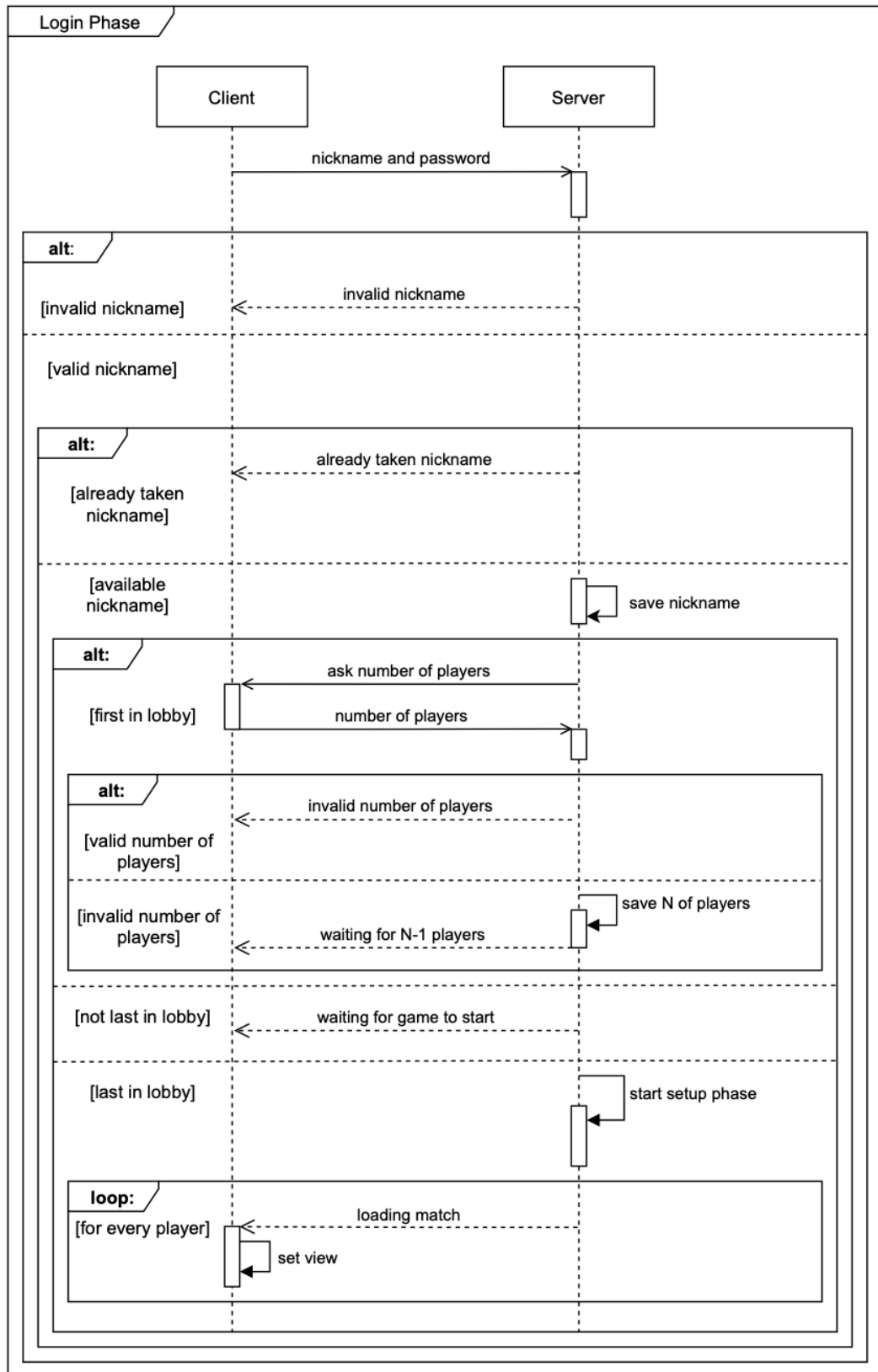
- TO SERVER: number of players

```
{
  "type": "lobbyChoice"
  "sender": "nick"
  "size": 2
}
```

- TO CLIENT: loading match (first view update)

```
{
  "type": "gameStarted"
  "players": ["nick1", "nick2", ..., "nickN"]
}
```

And It also send the first graphicUpdate with the fullGrid and fullMarket.



### 3 - Setup Phase

- **TO CLIENT: setup event (choice message)**

```

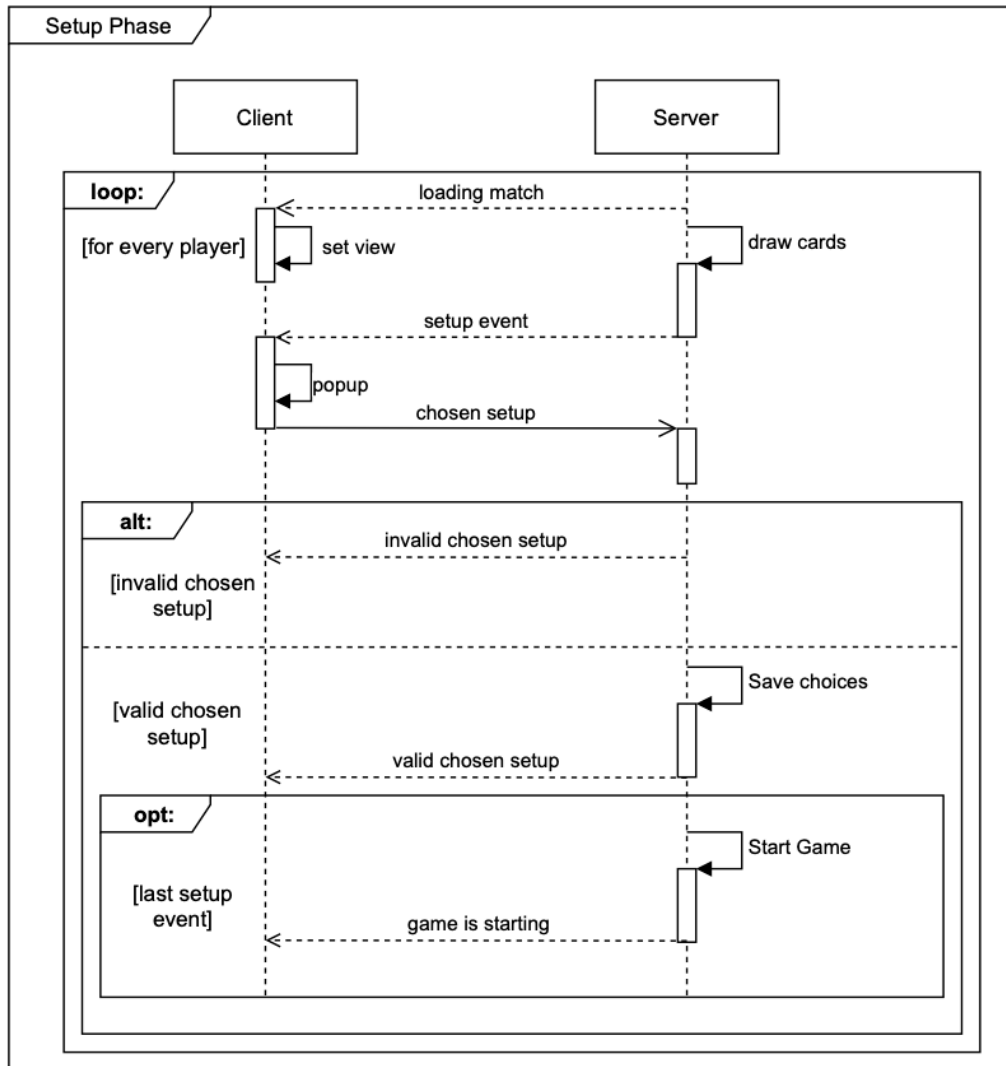
{
  "type": "setup"
  "leaderCardIDs": ["p1", "m1", "w3", "d2"], (leader cards to choose)
  "numberOfResources": 2 (number of resources to choose)
}
  
```

- **TO SERVER: chosen setup (action)**

```
{
  "type": "setupAction"
  "sender": "nick"
  "chosenLeaderCardIndexes": ["m1", "w3"], (chosen leader cards)
  "chosenResources": ["blue", "gray"] (chosen resources)
}
```

- **TO CLIENT: invalid/valid chosen setup**

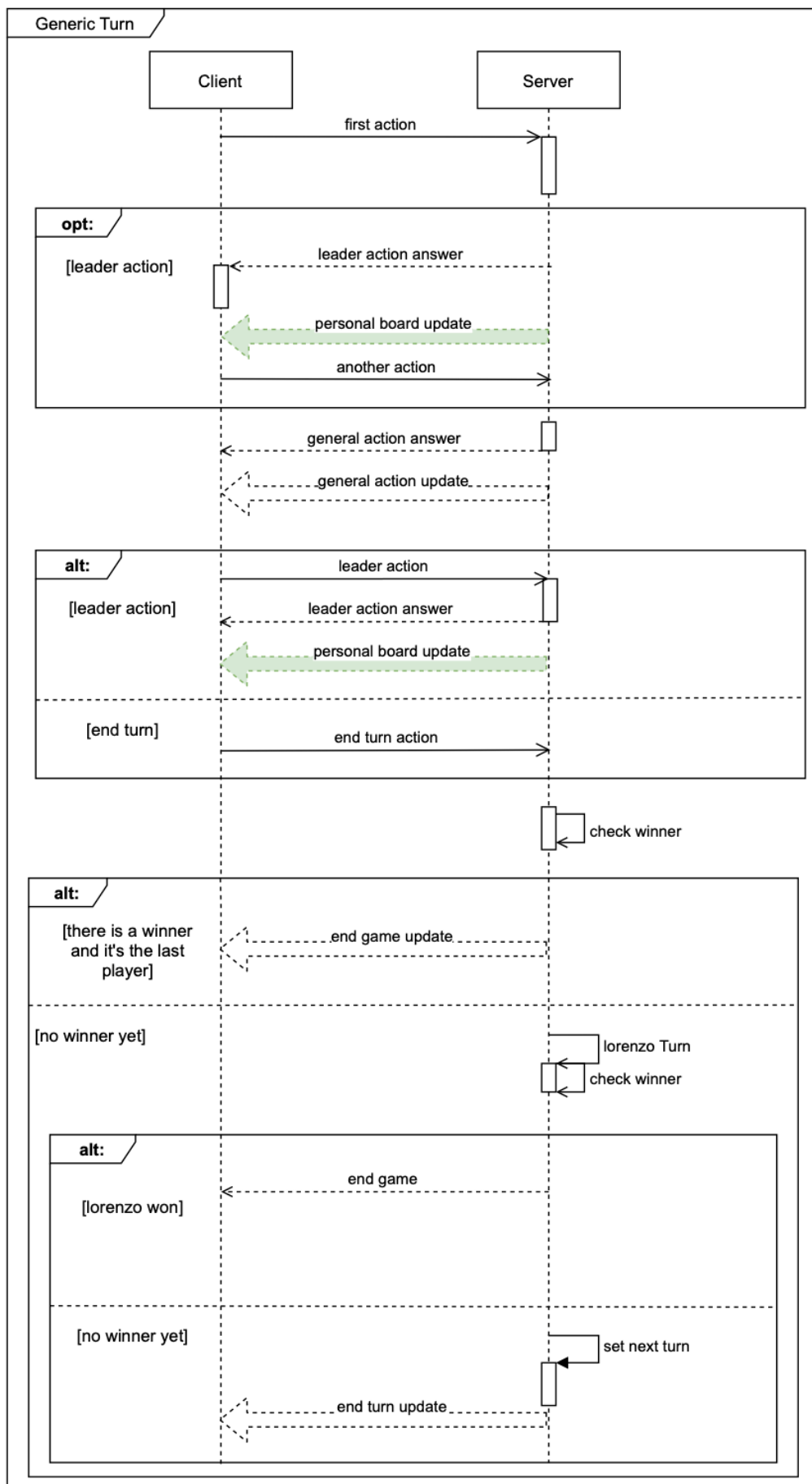
It's a **string message** containing the result of the *action* (error/success)



- **TO CLIENT: game is starting:**

It's a **view update message** with only the `"personalBoardUpdateList"` that contains all players' personal informations.

## 4 – Generic Turn (with Leader Action)



- **TO CLIENT: leader action answer & general action answer**  
It's a **string message** containing the result of the *action* (error/success)

- **TO SERVER: leader action**

It's an **action** with these parameters:

```
{
  "sender": "nickname",
  "type": "leaderAction",
  "card": "id",
  "discard": true (false to activate)
}
```

- **TO CLIENT: personal board update**

It's a **view update message** containing only the *"personal"* parameter

```
{
  "type": "graphicUpdate",
  "personal": [
    {
      "player": "nickname", (who did the leader action)
      "handLeaders": ["id1", "id2"],
      "activeLeaders": ["id1", "id2"],
      "productionBoard": ...
    }
  ]
  "faithTracksUpdate": ...
}
```

- **TO SERVER: end turn action**

It's an **action** with no extra parameters:

```
{
  "sender": "nickname",
  "type": "endTurnAction"
}
```

- **TO (all) CLIENTs: start turn update**

```
{
  "type": "startTurn"
  "nextPlayer": "nickname"
}
```

- **TO (all) CLIENTs: end game update**

```
{
  "type": "endGame",
  "results": [
    {
      "player": "nicknameWinner",
      "points": 100
    },
    ...
    {
      "player": "nicknameLast",
      "points": 10
    }
  ]
}
```

## 5 – Buy Action

- **TO SERVER: buy action**

```
{
  "sender": "nickname",
  "type": "buyAction",
  "cardLevel": 2,
  "cardColor": "green",
  "resourcesPositions": [4, 6, ..., 7]
}
```

- **TO CLIENT: placement choice**

It's a **choice message** containing the ID of the *DevelopmentCard* to place

```
{
  "type": "placeDevCard",
```

```

    "newCardID": "id"
  }
}

```

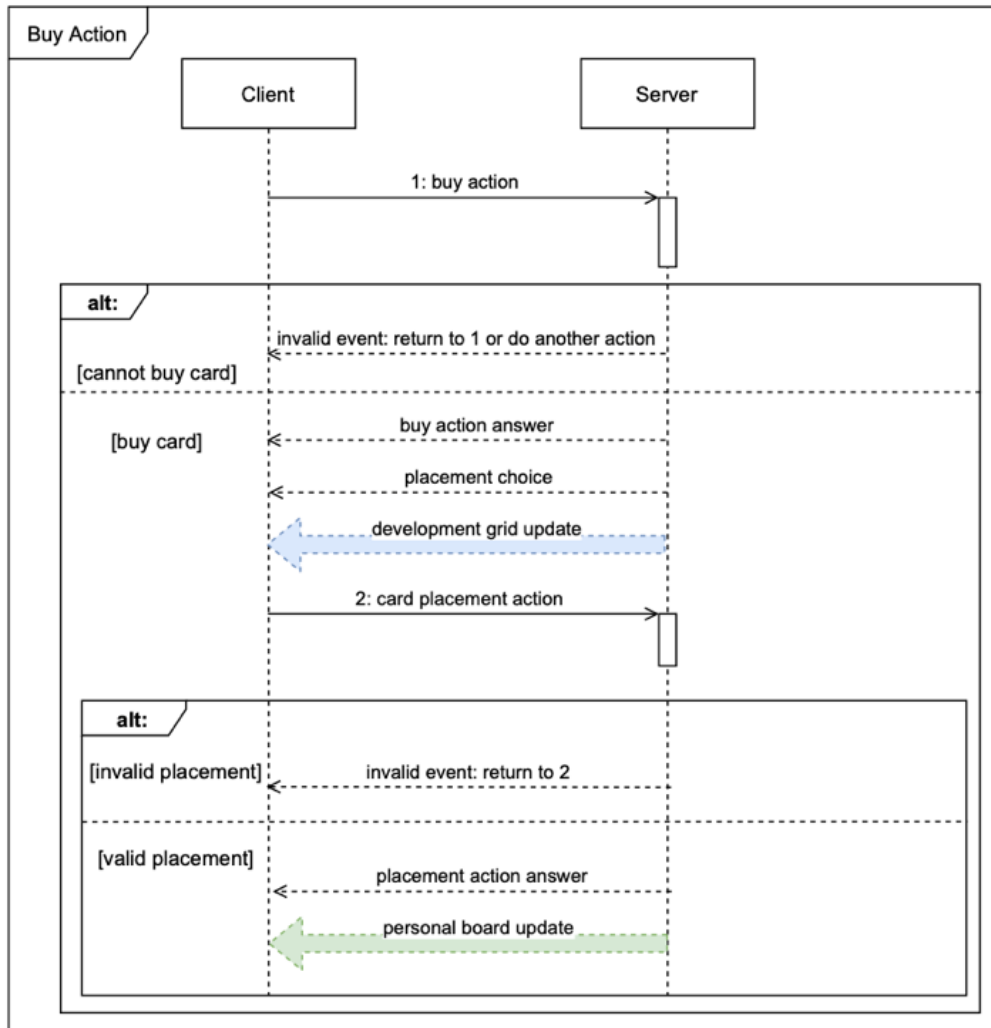
- **TO (all) CLIENTs: development grid update**

It's a **view update message** containing the new card to add to the *DevelopmentGrid*

```

{
  "type": "graphicUpdate",
  "gridUpdate":
  {
    "level": 2,
    "color": "yellow",
    "newCard": "id"
  }
  "personalBoardUpdateList": ...
}

```



- **TO SERVER: card placement action**

```

{
  "sender": "nickname",
  "slotPosition": 1
}

```

- **TO CLIENT: invalid event(s) & buy action answer & placement action answer**

They are **string message(s)** containing the result of the *action* (error/success)

- **TO (all) CLIENTs: personal board update**

It's a **view update message** containing the new personal board of the player who did the *action*

## 6 – Market Action

- **TO SERVER: market action**

```

{
  "sender": "nickname",
  "type": "marketAction",

```

```

    "arrowID": 2
  }
}

```

- **TO SERVER: resources placement action**

```

{
  "sender": "nickname",
  "type": "resourcesPlacementAction",
  "swaps": [0, 6, ..., 1, 10] (x2n = initial position, x2n+1 = final position)
  "isFinal": true (false to do more swaps after)
}

```

- **TO CLIENT: transformation choice & resources placement choice**

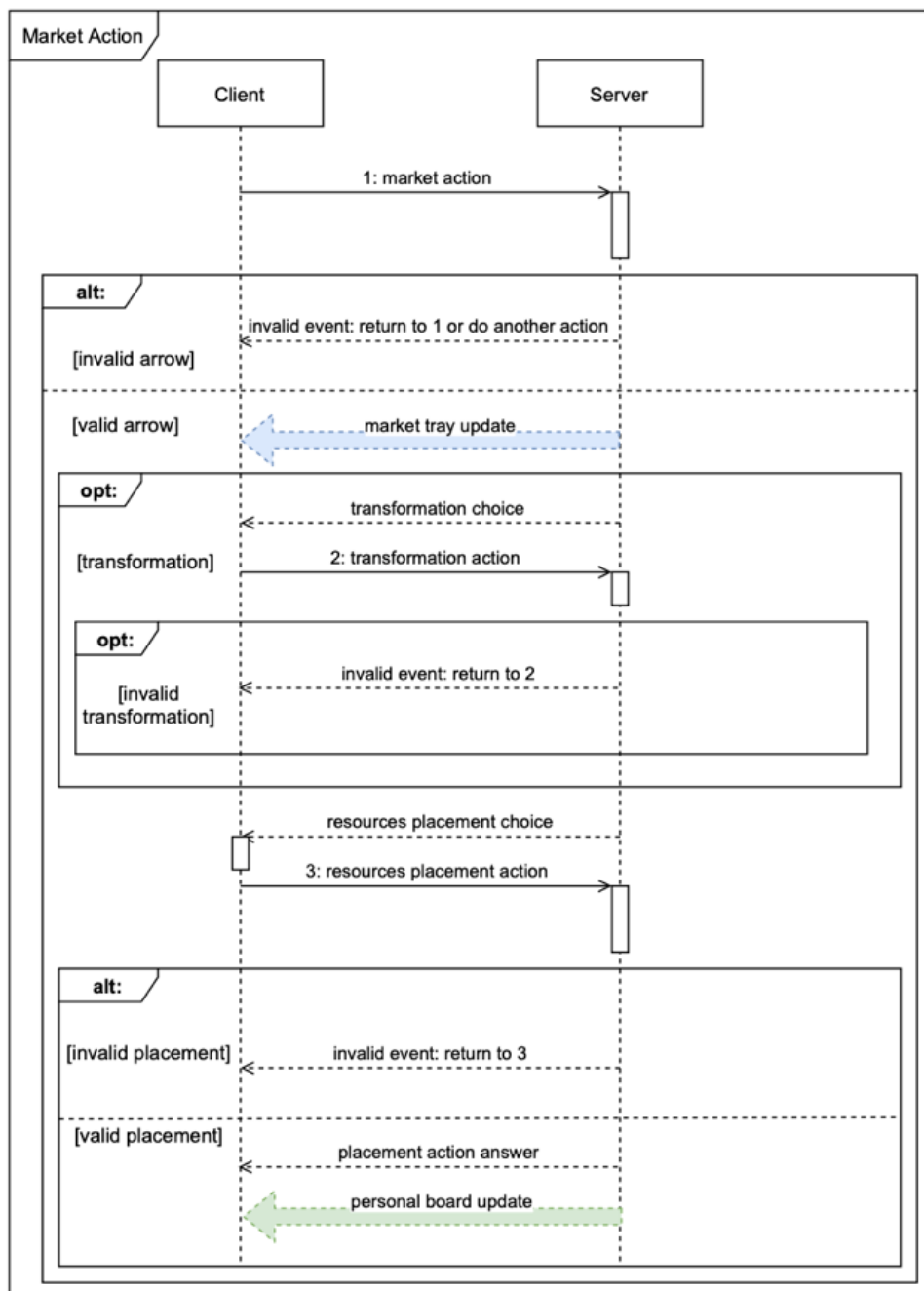
They are **choice message(s)** containing respectively the transformations to apply and the resources to place (as described in paragraph 3 of page 1)

- **TO (all) CLIENTs: personal board update & market tray update**

They are **view update message(s)** containing respectively the new personal board of the player who did the *action*, and the new *market tray*'s state

- **TO CLIENT: invalid event(s) & placement action answer**

They are **string message(s)** containing the result of the *action* (error/success)





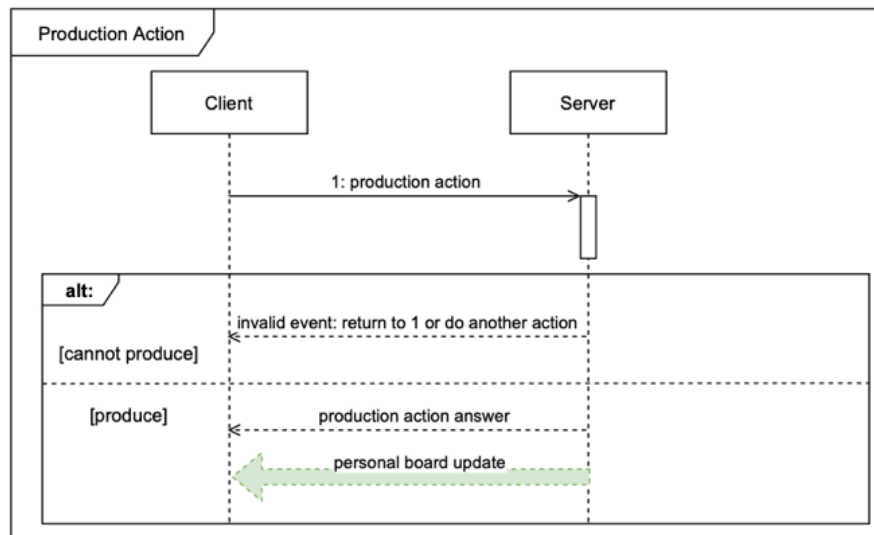
## 7 – Production Action

- **TO CLIENT: invalid event(s) & production action answer**  
They are **string message(s)** containing the result of the *action* (error/success)
- **TO (all) CLIENTs: personal board update**  
It's a **view update message** with the new personal board (*Warehouse* and *FaithTrack*) of the player who did the *action*
- **TO SERVER: production action**

```

{
  "sender": "nickname",
  "type": "productionAction",
  "inResForEachProd": [0, [2, ..., 7], ..., 3, [9, ..., 17]] (x2n = slot (key), x2n+1 = resources position)
  "outResForEachProd": [0, "blue", ..., 3, null] (y2n = x2n = slot (key), x2n+1 = desired resource)
}

```



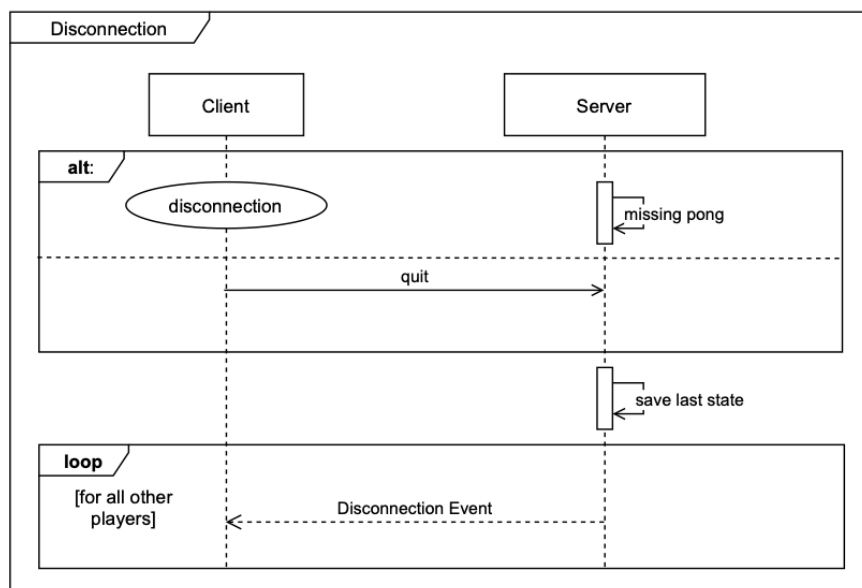
## 8 – Disconnection & Re-connection

During the *connection/login phase*, disconnections of the client aren't handled: it must try reconnecting to the server.

During the game, an unintentional disconnection is detected via a *ping* system, described as follows:

On the **server-side**, every 5 seconds the server sends a *ping* message to the client and the server starts a timer. If a pong answer isn't received before the timer ends the player's state is set as *disconnected* and its state is saved.

On the **client-side**, after receiving the first *ping*, an automated pong message is sent by the client which starts a timer of 8 seconds expecting another *ping* request. If it's not received, the server is considered disconnected and the application shuts-down with a notification.



If all the players are disconnected the application shuts-down.

- **TO SERVER: quit**

```
{  
  "type": "quit"  
}
```

- **TO (all) CLIENTs: disconnection event**

```
{  
  "type": "info"  
  "payload": "nickname has left"  
}
```

A player can reconnect to the game after an unintentional (or intentional) disconnection by logging-in using the same username and password inserted the first time. His turn will start from where he left.

- **TO SERVER: reconnection event**

Login message

- **TO CLIENT: reconnection update**

It's a **loading match** (loads the *MarketTray* and the *DevelopmentCard Grid*) fused with a **view update message** (loads all the players' personal boards)

- **TO (all) CLIENTs: notify reconnection**

```
{  
  "type": "info"  
  "payload": "nickname rejoined"  
}
```

