Politecnico di Milano - Prova Finale di Ingegneria del Software (AA 2021)
Group: AM65 – Beltrante Matteo, Bendinelli Marco, Bosisio Andrea

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.

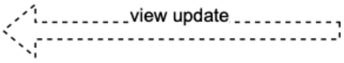
Some similar types of messages are used in different phases, so, in order to distinguish them, we defined the following common JSON scripts, characterized by different types:

FROM SERVER TO CLIENT:

1. **string message:**

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

2. **view update message:**
This kind of messages are sent to every client in order 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": "viewUpdate",
  "market": ["red", "yellow", "white", "green", "blue", … , yellow"],  (first: extra slot)
  "grid":
  {
    "level": 2,
    "color": "yellow",
    "newCard": "id"   ("empty" if empty slot)
  },
  "personal": [
    {
      "player": "nickname",
      "handLeaders": ["id1, id2"],
      "activeLeaders": ["id1, id2"],
      "productionBoard": [
          {
                "slot": 2,
                "newCard": "id"
          }
      ],
      "warehouse": [
        {
          "slot": 5,
          "type": "blue"
        },
        {
          "slot": 20,
          "type": "X"  ("X" if that slot is now empty)
        }
          … (for all resources)
      ],
      "faith": 0  (last marker update)
    }
  ]
}
```

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",  (constant type)

  "numberTransformation": 2,
  "possibleTransformation": ["blue", "green"],
or
  "resourcesPlacement": ["green", "yellow", "blue"],
or
```

```
    "cardPlacement": "id"
  }
```

FROM CLIENT TO SERVER:

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

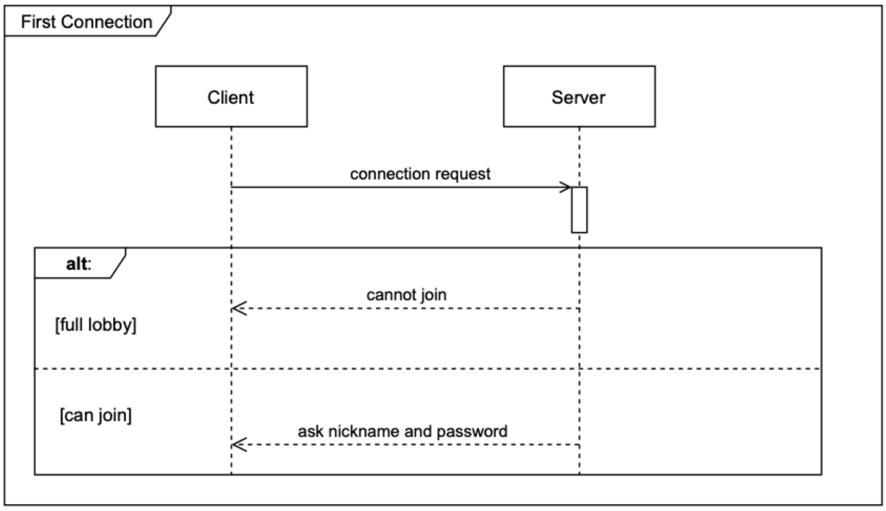
```
{
  "sender": "nickname",
  "type": "leaderAction, buyAction … ",

  "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 `"The Lobby is Full: connection has been denied"`.

2 - Login Phase

- **TO SERVER: nickname and password**

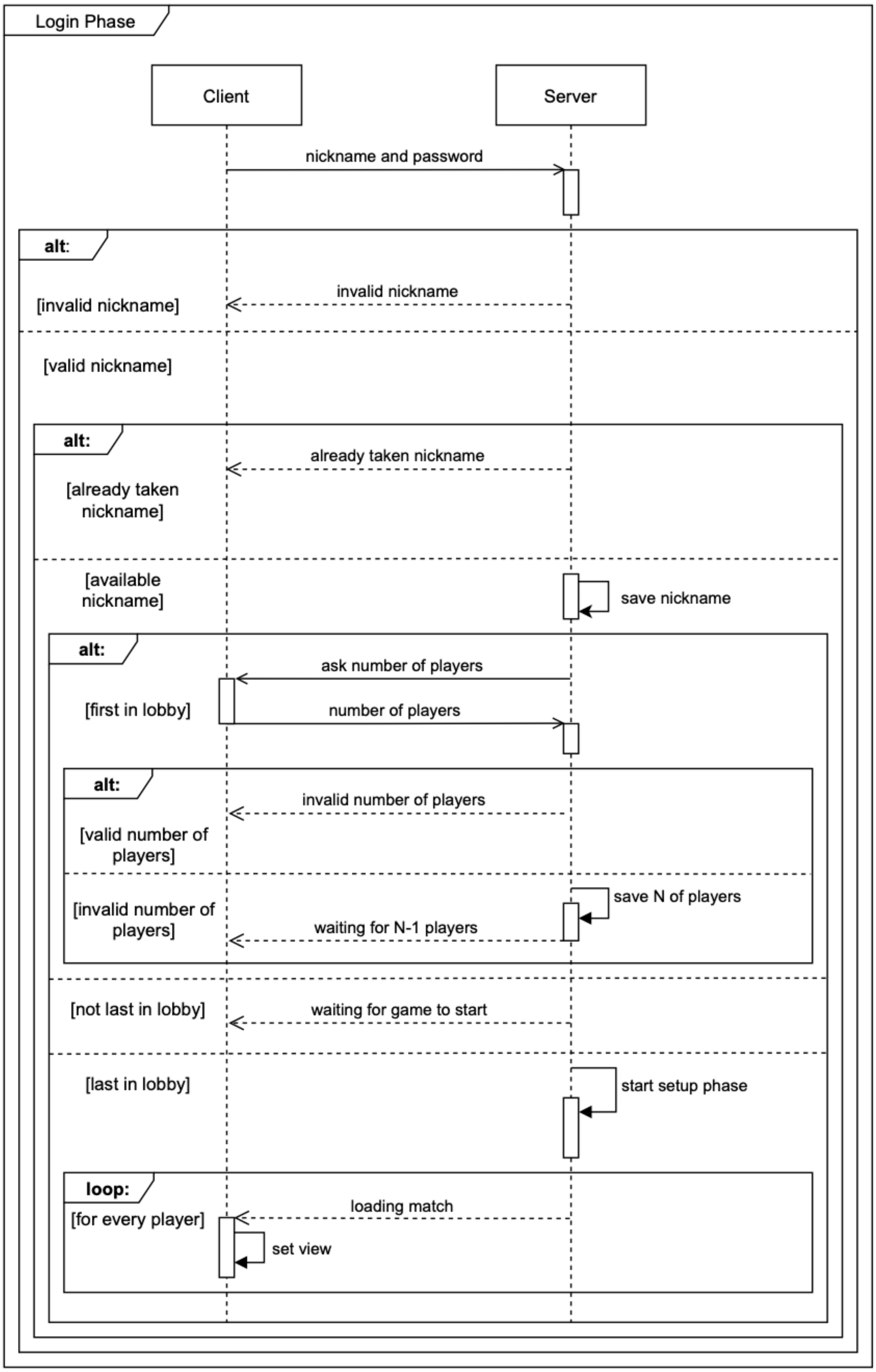
```
{
  "login": ["nickname","password"]
}
```

- **TO SERVER: number of players**

```
{
  "numberOfPlayers": 2
}
```

- **TO CLIENT: loading match** (first view update)

```
{
  "players": ["nick1", "nick2"", … , "nickN"],
  "market": ["red", "yellow", "white", "green", "blue", … , yellow"],  (first: extra slot)
  "grid": ["id1", "id2", … , "idEmpty"]  (only viewable layer)
}
```



3 - Setup Phase

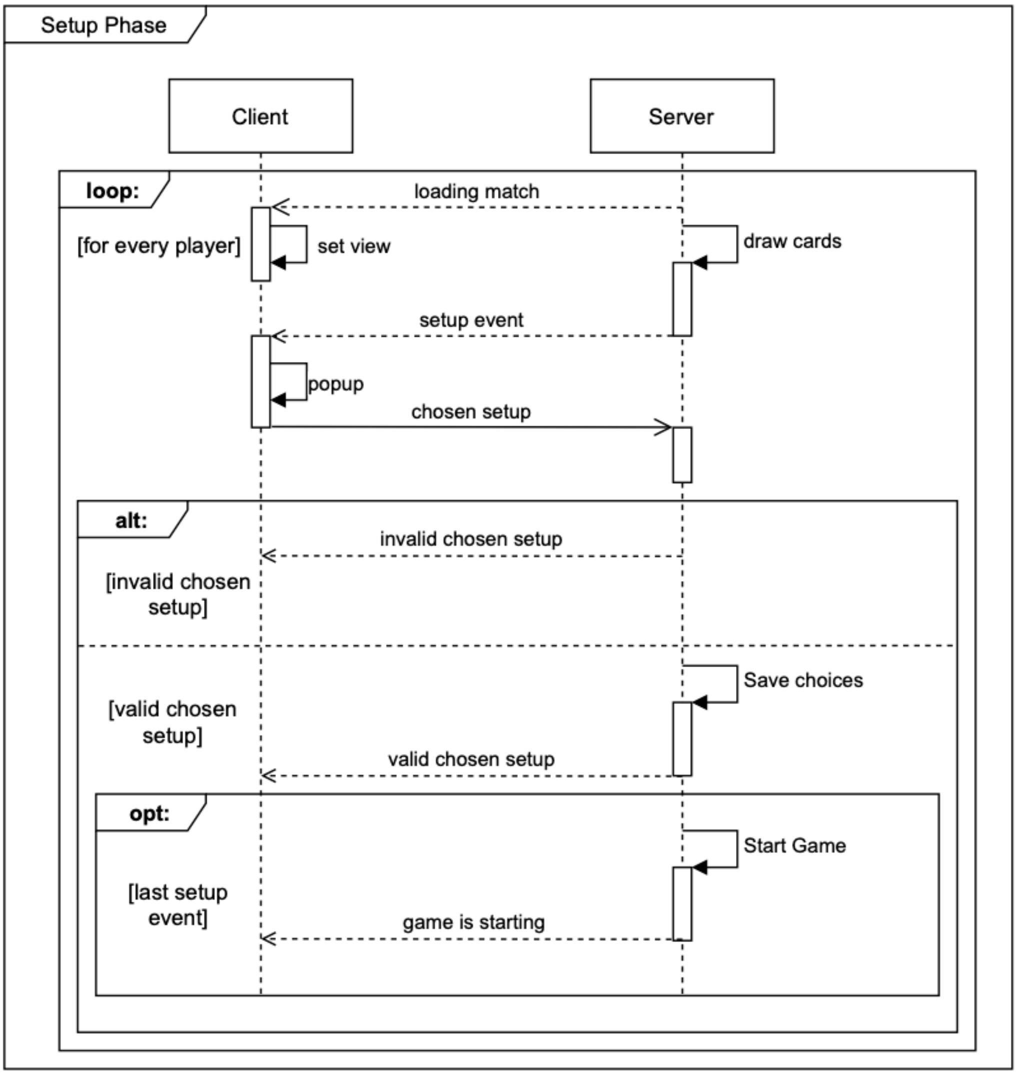
- **TO CLIENT: setup event**

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

- **TO SERVER: chosen setup**

```
{
  "leaders": ["m1", "w3"],  (chosen leader cards)
  "resources": ["B", "G"]  (chosen resources)
}
```

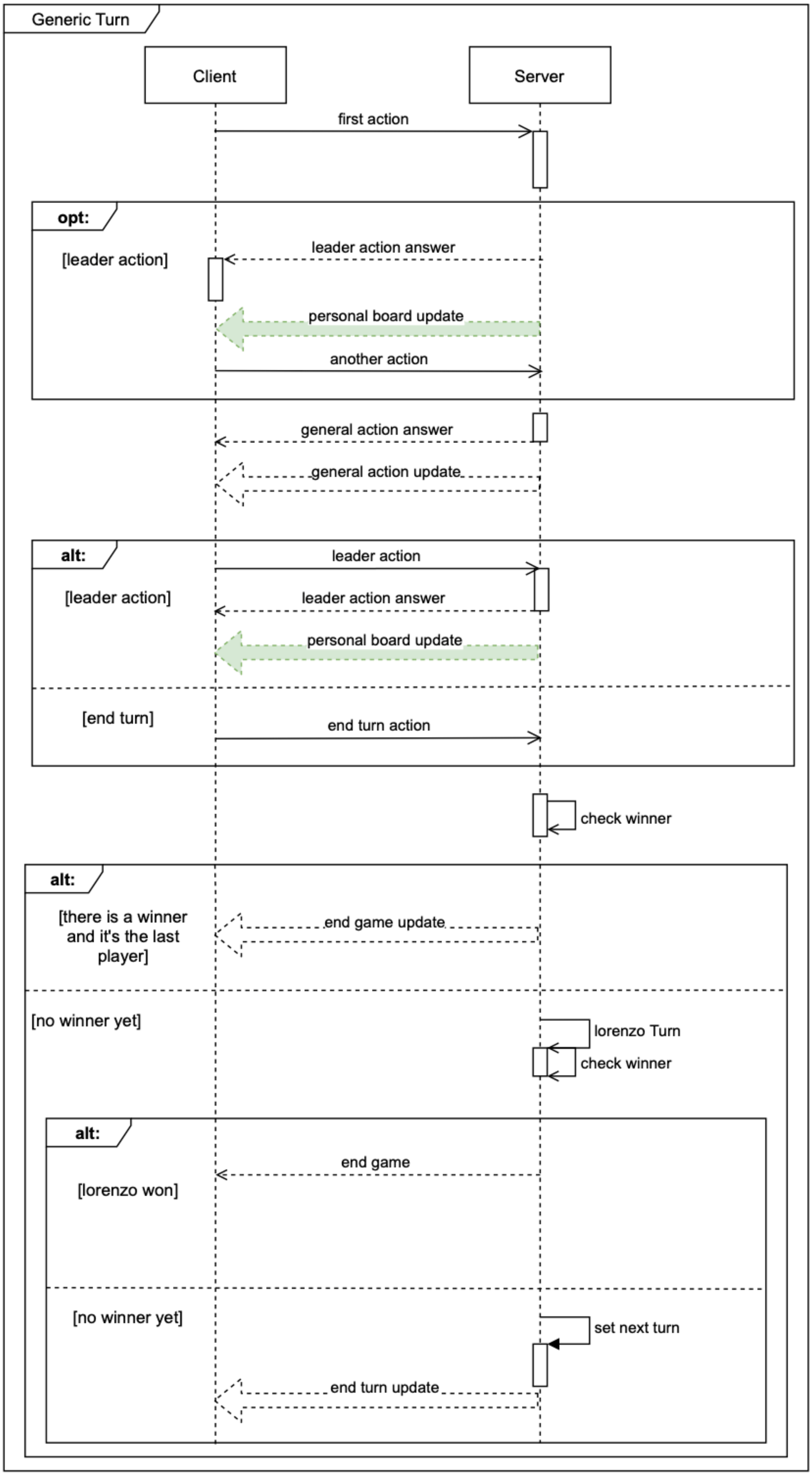
- **TO CLIENT: invalid/valid chosen setup**
  It's a **string message** containing the result of the *action* (error/success)



- **TO CLIENT: game is starting:**
  It's a **view update message** with only the *"personal"* parameter

```
{
  "type": "viewUpdate",
  "personal": [
    {
      "player": "nick1",
      "hand": ["m1, w3"],
      "warehouse": [
        {
          "type" : "B",
          "position" : 5
        },
  … (for all chosen resources)
      ],
      "faith": 0
    },
… (for all the players)
  ]
```

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": "viewUpdate",
  "personal": [
    {
      "player": "nickname",  (who did the leader action)
      "handLeaders": ["id1, id2"],
      "activeLeaders": ["id1, id2"],
      "productionBoard":
      "faith": 1  (if discard)
    }
  ]
}
```

- **TO SERVER: end turn action**
  It’s an **action** with no extra parameters:

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

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

```
{
  "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**

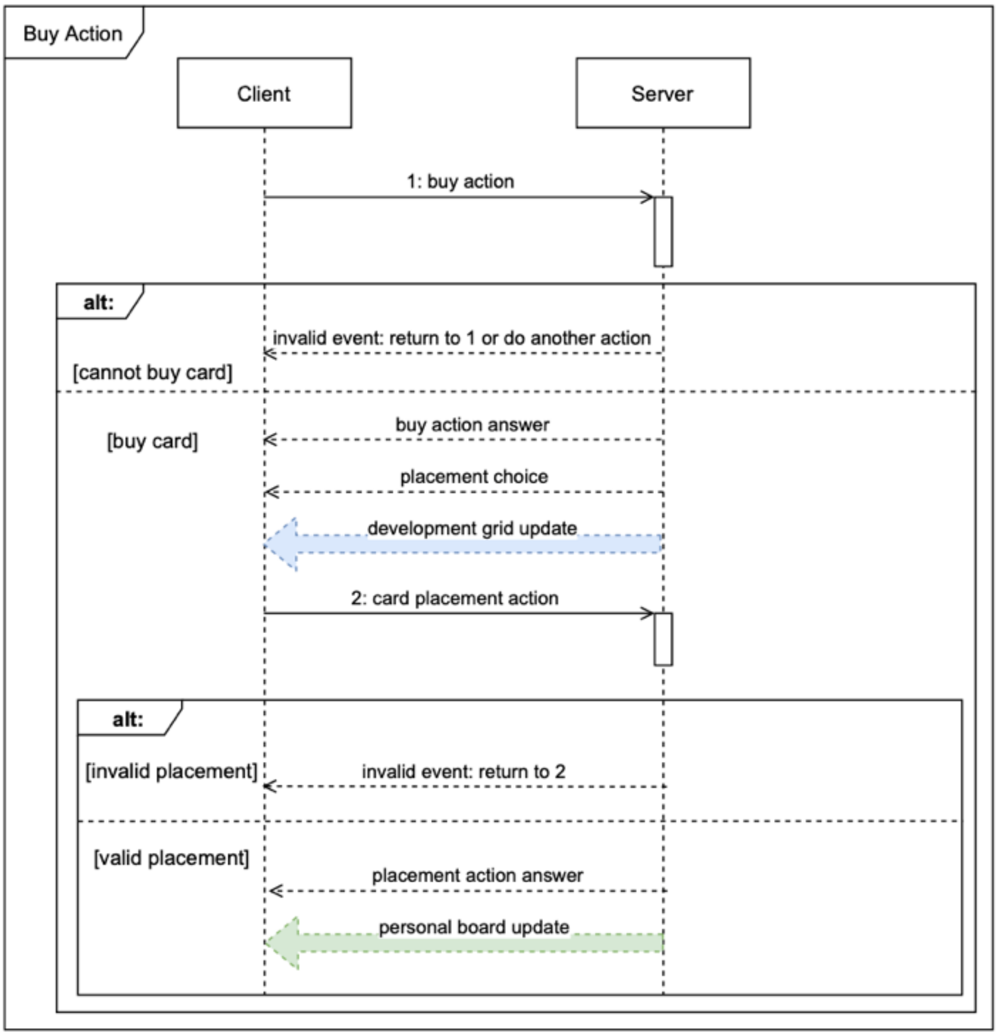
```
{
  "sender": "nickname",
  "type": "buyAction",
  "cardLevel": 2,
  "cardColor": "green",
  "resourcesPositions": [4, 6, … , 7]
}
```

- **TO CLIENT: placement choice**
  It’s a **choice message** containing the ID of the *DevelopmentCard* to place

```
{
  "type": "choice",
  "cardPlacement": "id"
}
```

- **TO (all) CLIENTs: development grid update**
  It's a **view update message** containing the new card to add to the *DevelopmentGrid*

```
{
  "type": "viewUpdate",
  "grid":
  {
     "level": 2,
     "color": "yellow",
     "newCard": "id"
  }
}
```



- **TO SERVER: card placement action**

```
{
  "sender": "nickname",
  "type": "cardPlacementAction",
  "slot": 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",
  "arrow": 2
}
```

- **TO SERVER: resources placement action**

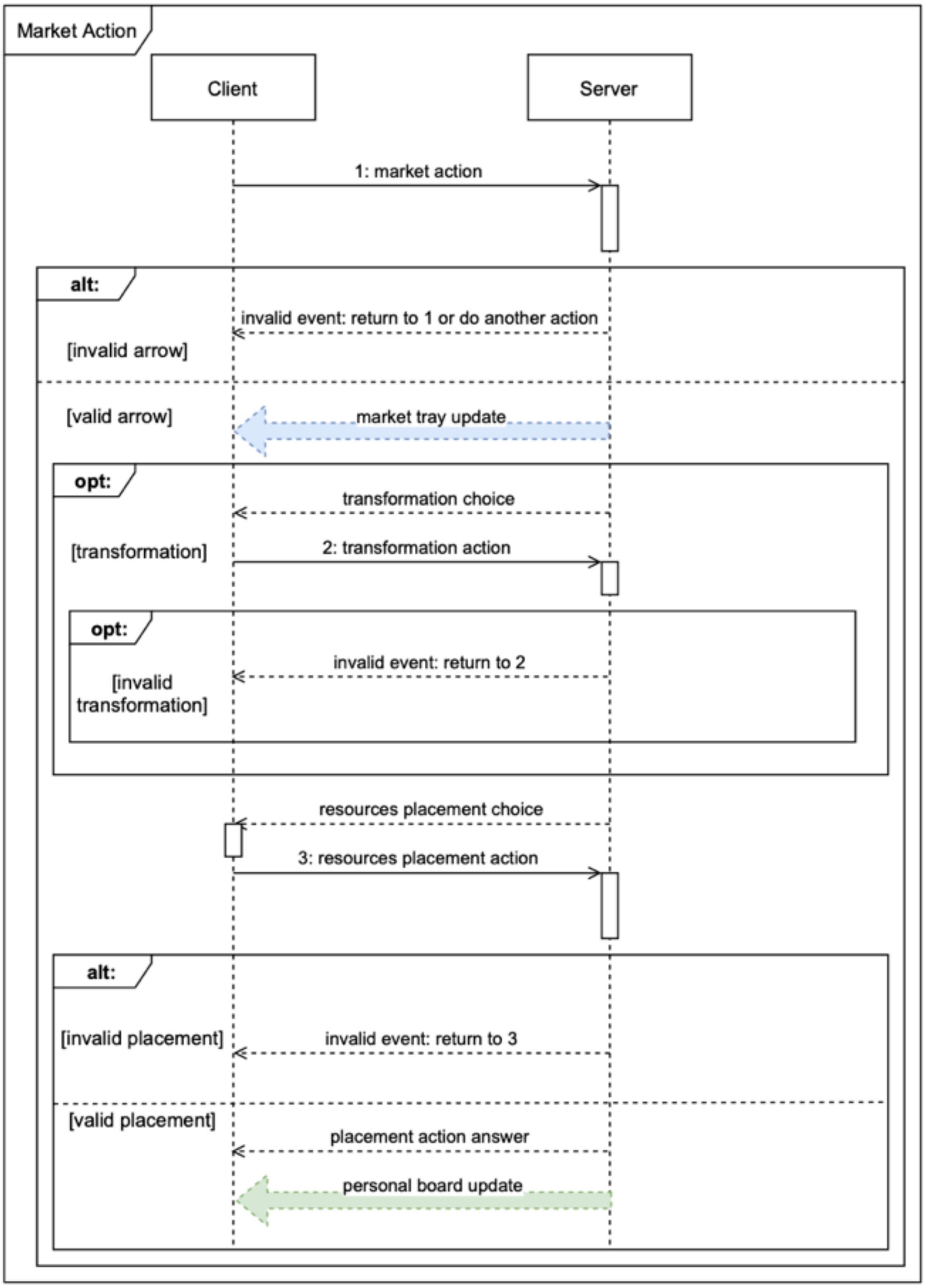
```
{
  "sender": "nickname",
  "type": "resourcesPlacementAction",
  "swaps": [0, 6, … , 1, 10]
}
```

(x$_{2n}$ = initial position, x$_{2n+1}$ = final position)

- **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)



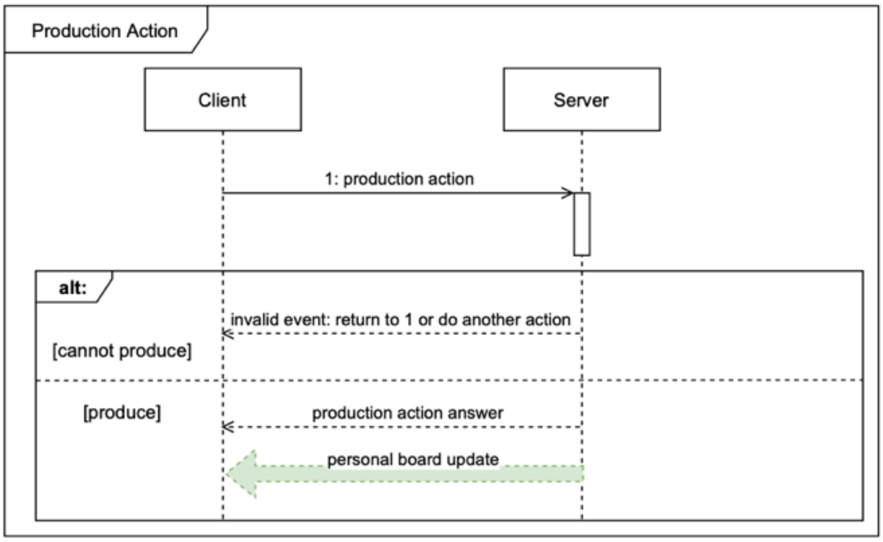
7 – Production Action

- **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*

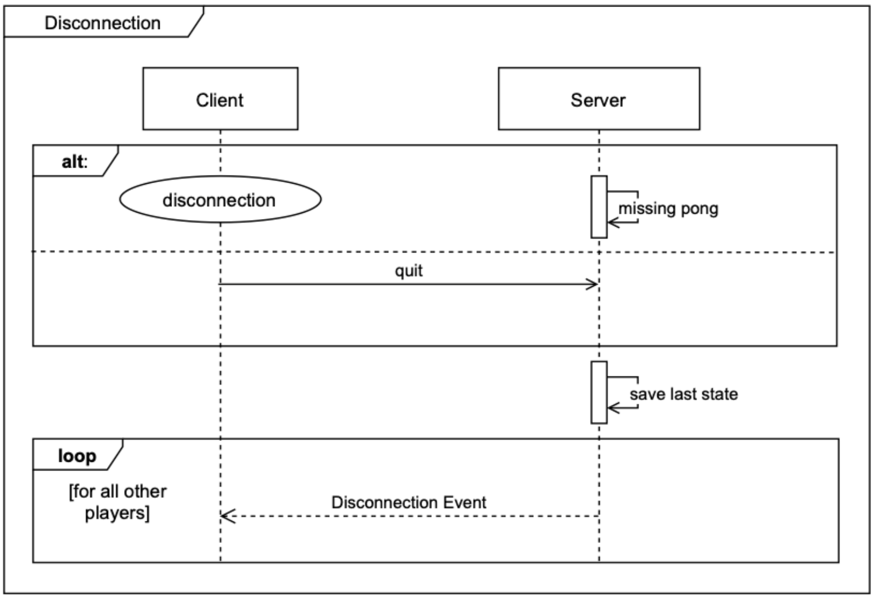
- **TO SERVER: production action**

```
{
  "sender": "nickname",
  "type": "productionAction",
  "productionIn": [0, [2, … , 7], … , 3, [9, … , 17]] (x2n = slot (key), x2n+1 = resources position)
  "productionOut": [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 has to 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 20 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 40 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**

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

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

```
{
  "quitter": "nickname"
}
```

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.

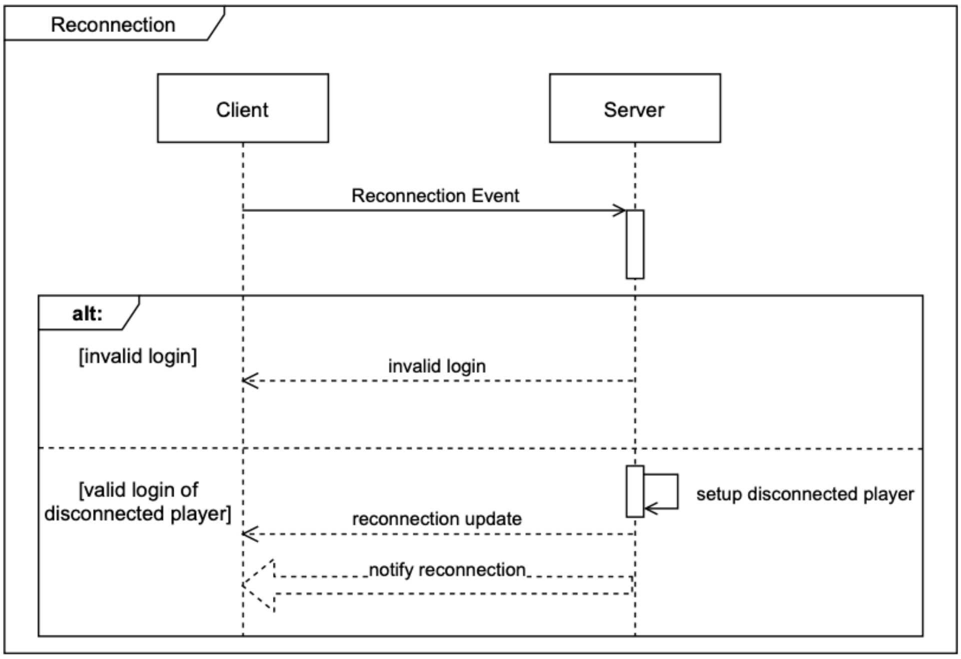
- **TO SERVER: reconnection event**

```
{
  "login": ["nickname", "password"]
}
```

- **TO CLIENT: reconnection update**
  It's a **loading match** (loads the *MarketTray* and the *DevelopmentCard Grid*) followed by a **view update message** (loads all the players' personal boards)

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

```
{
  "rejoiner": "nickname"
}
```


Reconnection
Client
Server
Reconnection Event
alt:
[invalid login]
invalid login
[valid login of disconnected player]
setup disconnected player
reconnection update
notify reconnection