Communication Protocol - AM28

Introduction

Our communication protocol is based on the idea that a Server is listening for messages sent by Client, so when the Server receives a message first of all checks if it is "legal", then executes the associated operation and replies to the Client with a message containing the succeed of the operation and the changes.

The communication is built on a single socket but the messages are divided into three "channels" created by sorting the messages received into differents VirtualSocket.

This idea bring to us two vantages:

- Make the updating View mechanism and the Client to server command indipendent
- Check the connection state of only one socket

The messages previously cited are instances of objects we call Packet that are serialized, sent, and once received by the server, deserialized using **JSON**.

Our packet contains:

- **Header**: containing the identifier of the packet.
- **Channel**: the channel of the packet.
- **Body**: a string containing an instance of a Body object, serialized with **JSON.**

The channels we choose to divide the socket are:

- PlayerAction: in which the player sends to the Server the command he wants to execute
- ViewNotify: in which the model notifies the changes to the View
- ConnectionStatus: in which the client ping the Server

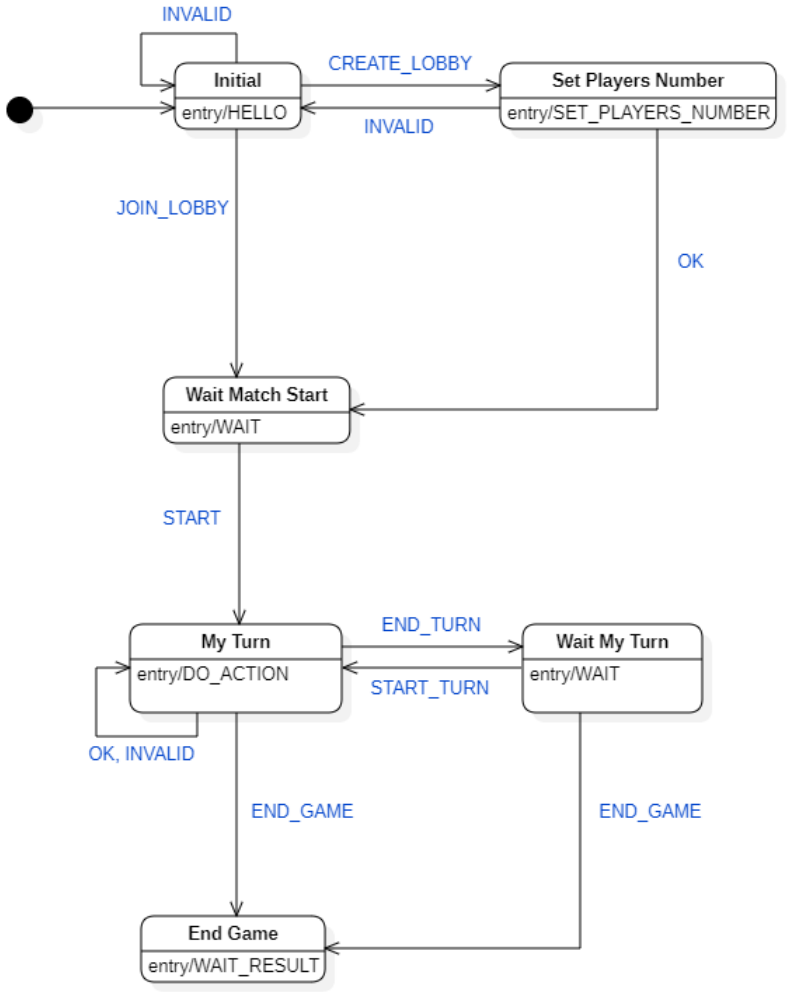
PlayerAction Channel

In this channel we choose to design the Client as a finite state automata that is based on:

- Server messages as input of the automata;
- Client messages as state of the automata;

So, whenever the Client enters a new state, it sends to the Server the associated message and waits for the response.

The response obtained is the input for changing the state of the player and sends a new message to continue the communication.



The communication is divided into two big phases:

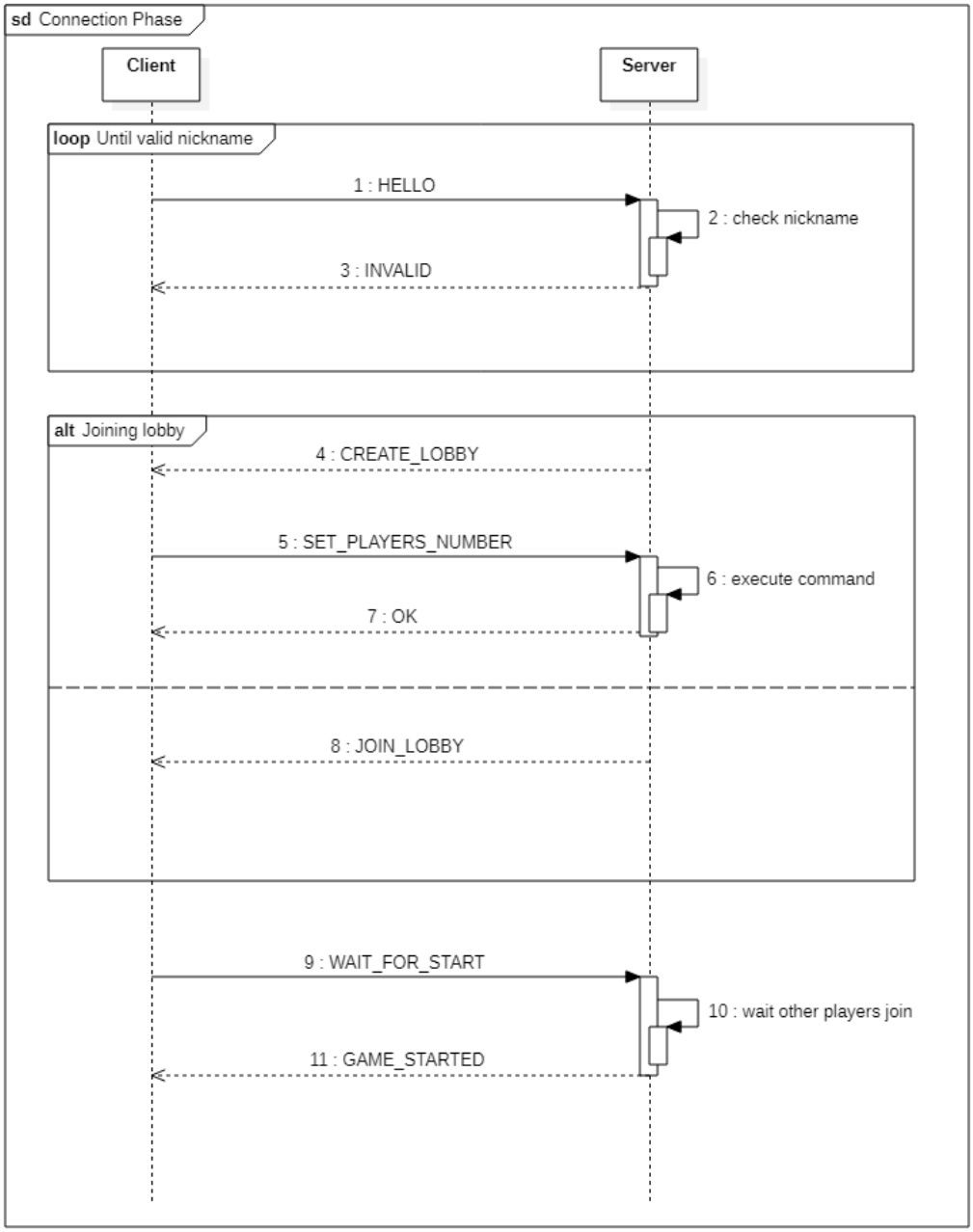
1. Connection Phase
2. In Game Phase

In both phases, the Server reacts to Client messages by executing the actions requested by the Client or sending an error message.

Connection Phase

The Connection Phase starts when a Client connects to the Server. The Client immediately sends the nickname of the player and the Server checks if the nickname is valid. If the nickname isn't valid, the Server asks again to the Client to insert another nickname, otherwise the Server puts the player in the Lobby.

If the player is the first one to connect, the Server creates a Lobby and the Client sends the number of player that will join the Match, otherwise the Server puts the player in a Lobby previously created. When all the players are connected, the Server starts the game.




```
{
  "Header": "HELLO",
  "Channel": "PLAYER_ACTION",
  "Body":{
    "Message" : "TheDarkVinz"
  }
}
```

```
{
  "Header": "SET_PLAYERS_NUMBER",
  "Channel": "PLAYER_ACTION",
  "Body":{
      "Command" : {
        "type": "SetNumberCommand",
        "number": 2}
    }
}
```

```
{
  "Header": "CREATE_LOBBY",
  "Channel": "PLAYER_ACTION",
  "Body": {
    "Message": "You are the first"
  }
}
```

In Game Phase

After the game started the Client can do an action during his turn, or waiting for it. The Server may answer in different ways based on the player's decision:

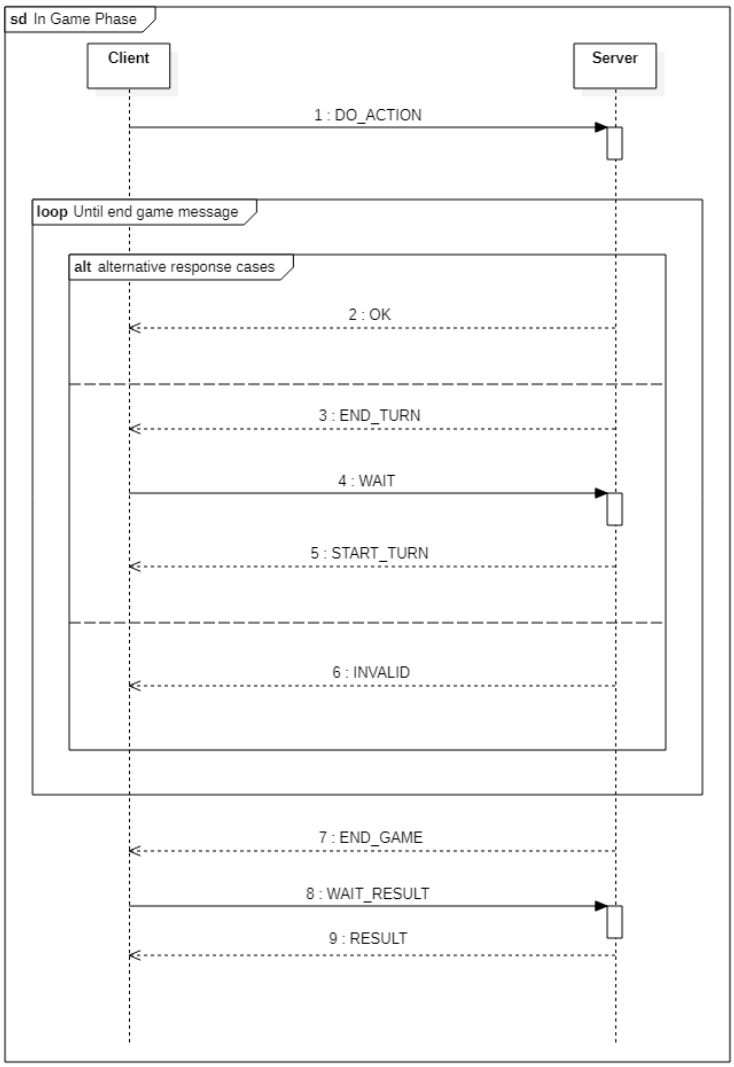
- OK if the action is completed, for example playing a Leader card, and the Client can do another action;
- END_TURN if the Client has done the main action and the Client must waits for his turn;
- INVALID if the Client tried to do an illegal move, he has to do an action.

In the **In Game Phase**, the set of messages that the Client can compose is limited by the View, indeed the player has a finite set of actions that he can perform at the beginning of the turn: the actions he cannot perform are not shown to the player.

Thinking about the additional function, if a client is disconnected from the Server, the match doesn't stop and the Server will just skip the player's turn. Even if the Client reconnects, the player won't recover any lost turn.

When the game's finished, the Server updates all Clients and changes their state in waiting for results. Shortly after, the Server updates again every Client with the leaderboard.

As soon as the game ends, it doesn't matter how, the connection between every remaining Client and the Server is interrupted.



```
{
  "Header": "DO_ACTION",
  "Channel": "PLAYER_ACTION",
  "Body": {
      "Command" : {
        "type": "BuyDevCard",
        "row": "LEVEL1",
        "col": "BLUE",
        "slotDestination": "LEFT"}
    }
}
```

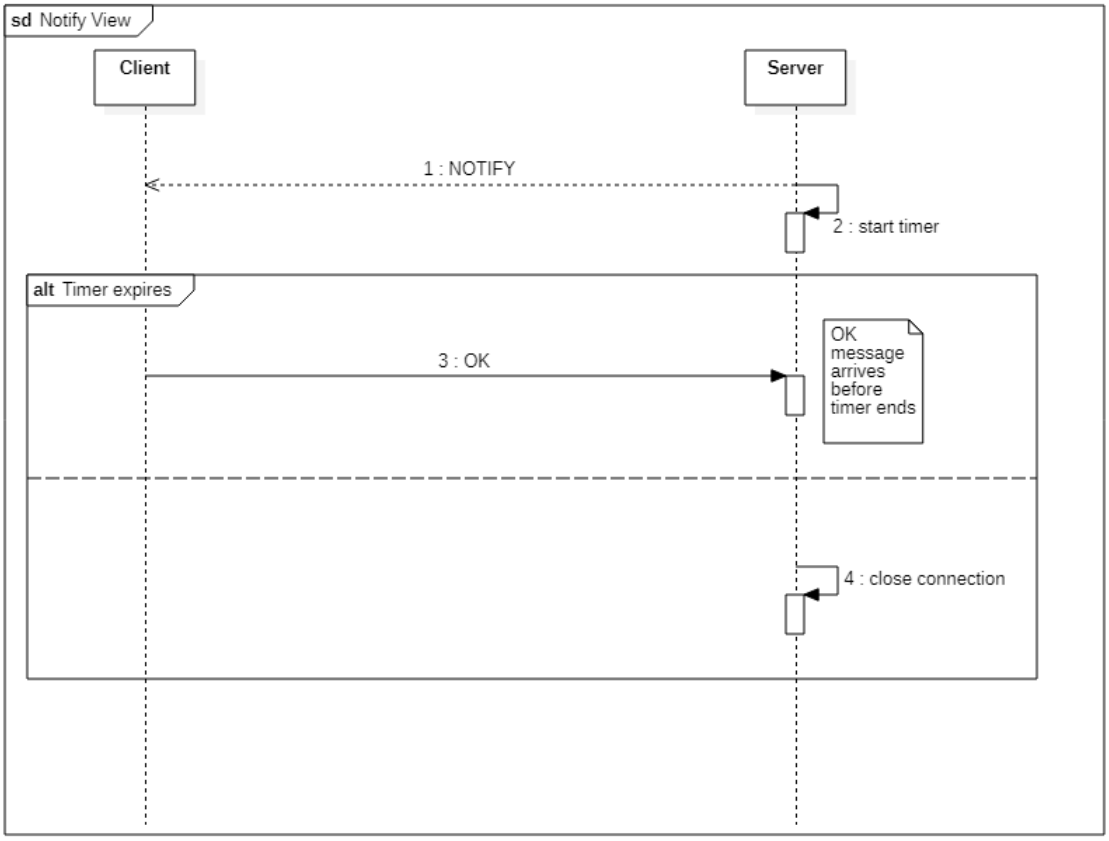
```
{
  "Header": "END_TURN",
  "Channel": "PLAYER_ACTION",
  "Body":{
    "Message" : "Your turn is over"
  }
}
```

```
{
  "Header": "INVALID",
  "Channel": "PLAYER_ACTION",
  "Body": {
    "Message": "You can't do this action",
    "Error": "PlayerStateException"
  }
}
```

```
{
  "Header": "RESULT",
  "Channel": "PLAYER_ACTION",
  "Body": {
    "Message" : "This is the Leaderboard: 1° MadCini, 2° LastBuddy, 3° lino, 4° TheDarkVinz"
  }
}
```

NotifyView Channel

This channel serves the purpose of updating the Client's view. Whenever the model is modified in a way that impacts the game, the Server will notify every active Client of the changes. The Client answer with an OK if the update is received, else the Server will send again the message after n seconds. If the Client doesn't answer again, the Server will close the connection and the Client must reboot if he wants to reconnect to the match.

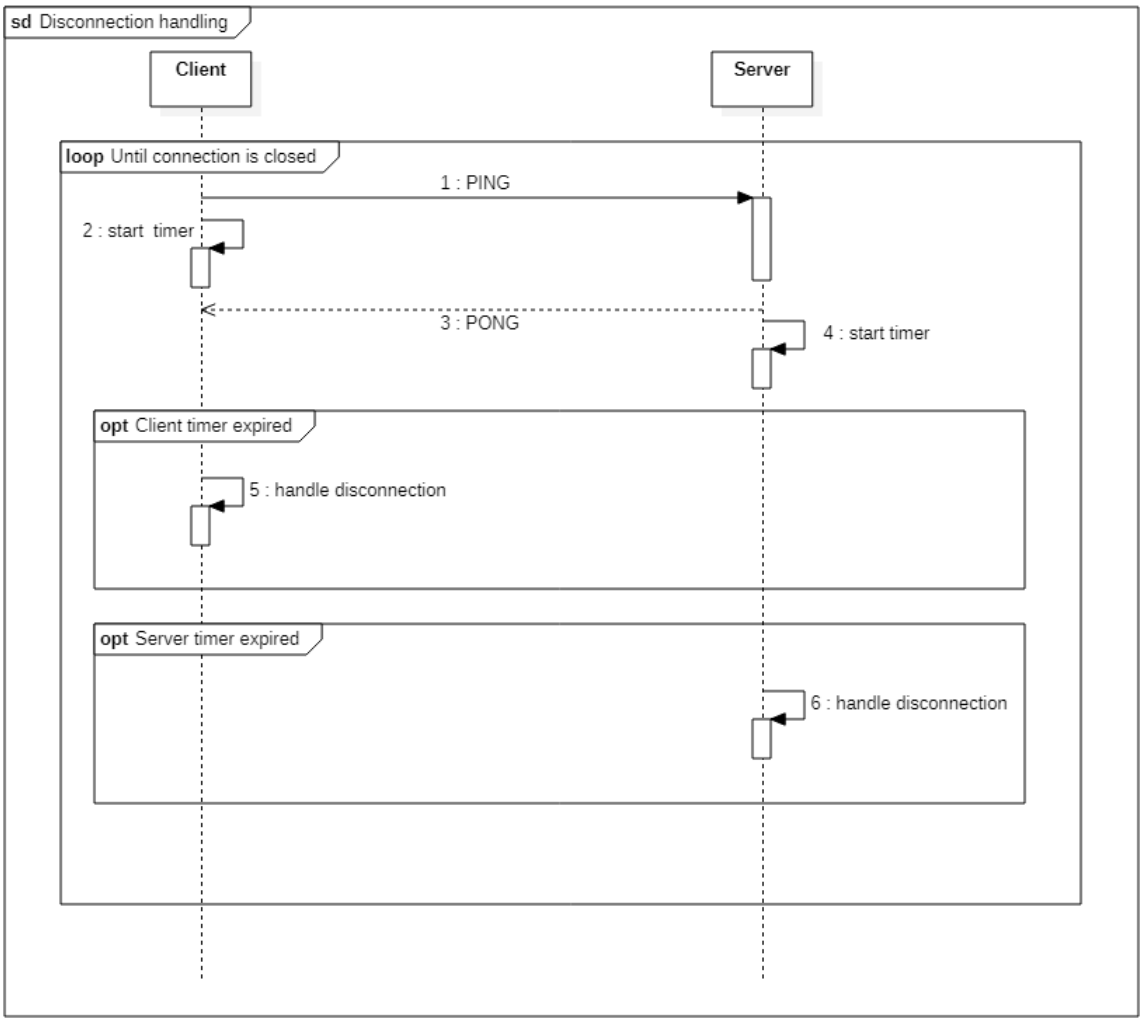



```
{
  "Header": "NOTIFY",
  "Channel": "NOTIFY_VIEW",
  "Body":{
    "field": "DepotSlot.BOTTOM",
    "newValues": [
      {
        "type": "COIN",
        "amount": 2
      }
    ]
  }
}
```

ConnectionStatus Channel

While connected, the client periodically sends a ping and expects a pong from the Server. The Server expects periodical pings from the Clients: if for some reason, the Client stops sending pings, the server will recognize the Client as disconnected.

During the **In Game Phase**, the player can decide at any given time to disconnect from the game. If the match is still ongoing, the player can reconnect back to the game identifying himself via nickname, using the same one as the first. If every player but one disconnects from the game, the Server will finish the match and the last player standing will win the game no matter what.




```
{
  "Header": "PING",
  "Channel": "CONNECTION_STATUS",
  "Body":{}
}
```

```
{
  "Header": "PONG",
  "Channel": "CONNECTION_STATUS",
  "Body":{}
}
```