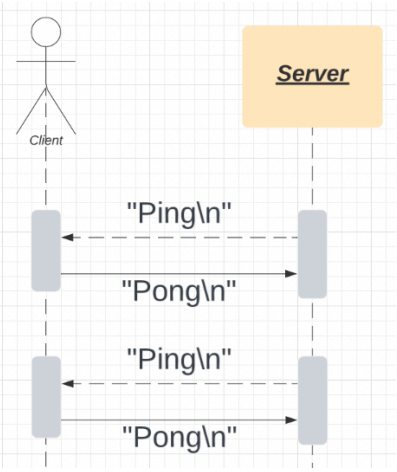
Communication Protocol

- **Connection Control**

When the connection is initialized the server sends a "ping\n" message (plain text) and the client replies with a "pong\n" message (plain text).
This procedure is performed every time a new client connects to the server. Also, when a new client connects, the server verifies that the others are still connected, sending them a "ping\n" message and waiting for a "pong\n" reply.
This messages are exchanged periodically in parallel (different thread) to the messages that the player will exchange with the Server, if the number of messages received in a period of time is insufficient the Server will signal the player as Inactive until a reconnection or a new “message is received.



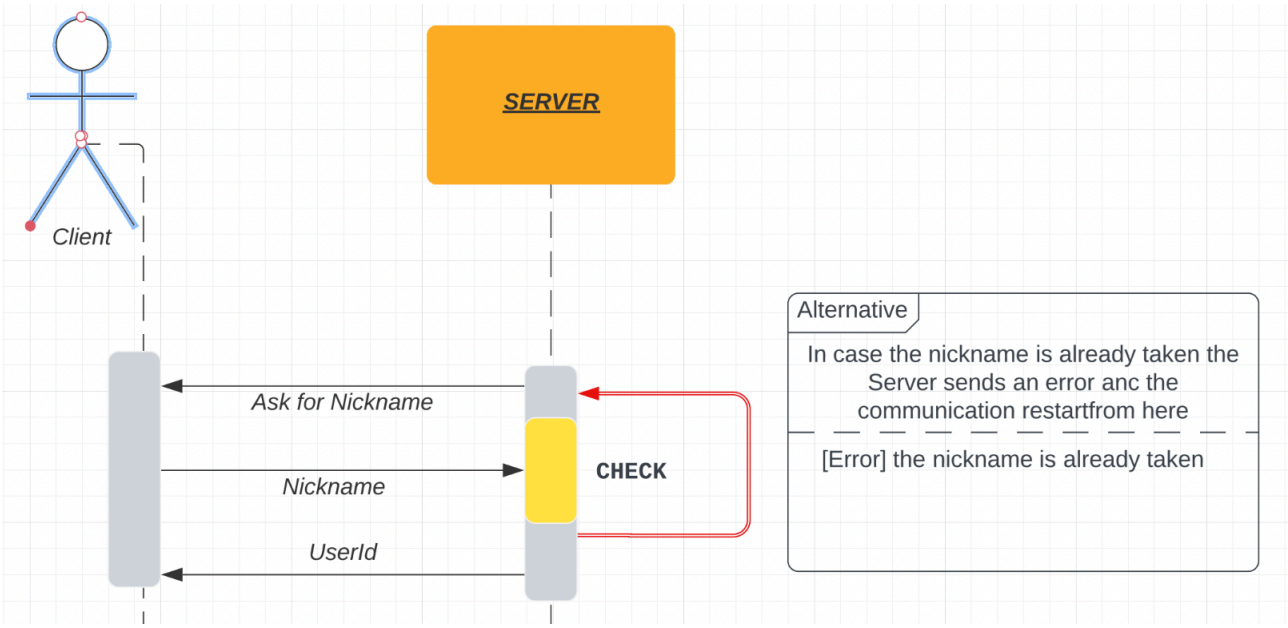
- **Actions**

During the game, the client may want to interact with the server in different phases of the match, the messages will follow the JSON protocol which allows for a smoother encoding and decoding, while maintaining an “easy to read” approach while analyzing the text.
The game actions can be divided in different phases:

A. Initialization phase
B. Game phase

Fase A: Initialization

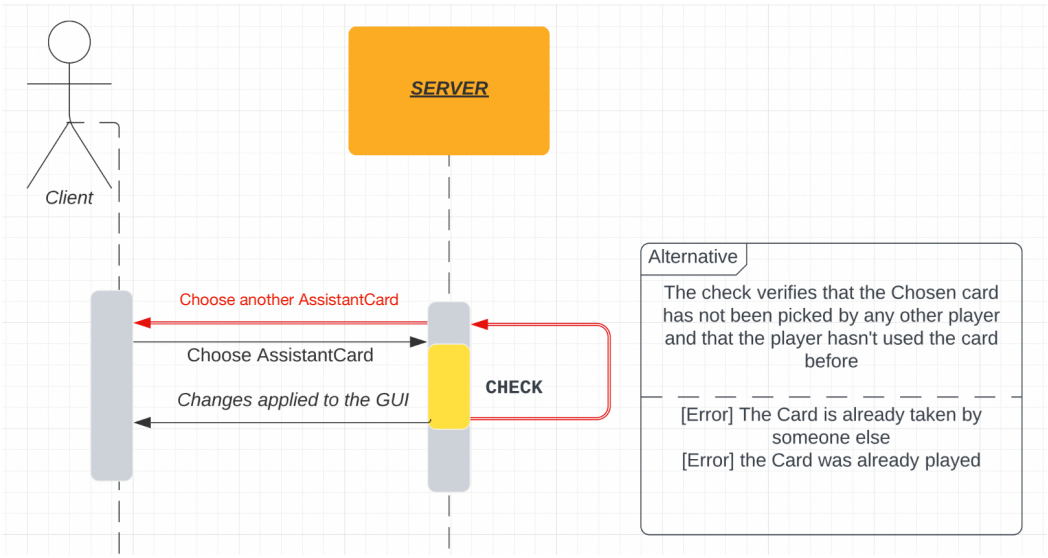
In the initialization phase the Client tries to login as a Player into the game. After receiving a positive “pong\n” message, meaning the client is still active, the server asks the client to insert a nickname; after the nickname is passed to the Server, a check occurs to see if the player already exists, if the check succeeds the player is assigned of a userId, otherwise the Server responds with an error asking to input a different username.



Fase B: Game actions

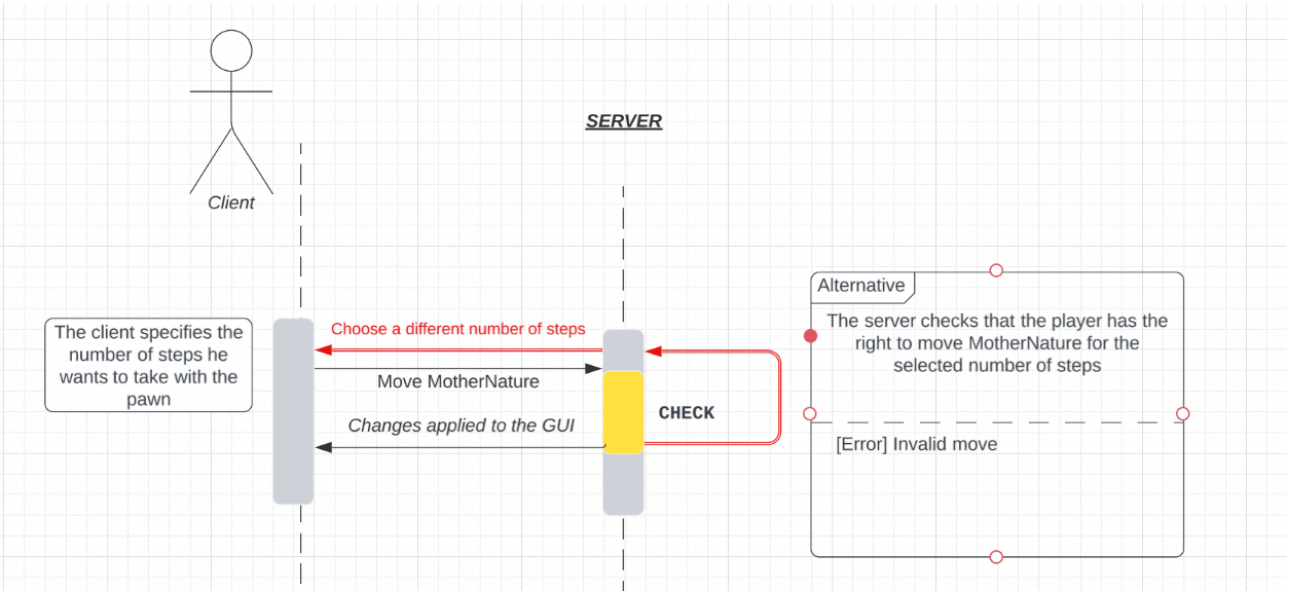
Choosing an Assistant Card

While checking the player is still active through “ping-pongs” when it’s the current player’s turn the server asks to choose an Assistant Card, the Client sends the number of the card he wants to choose and the Server executes a check to see if the move is “legal”, proven that, the server changes the state of the Game model and consequently the GUI the player is interacting with. In case the move is illegal the server sends back an error explaining the problem with the client’s request.

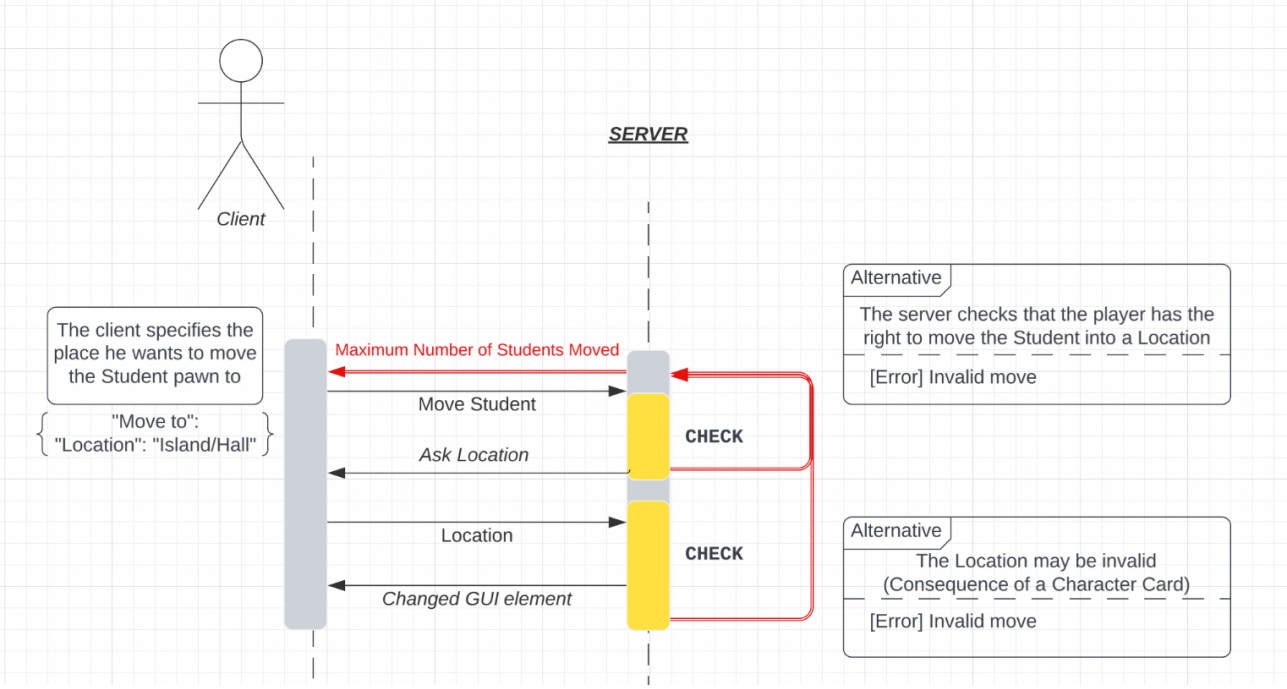


Moving the MotherNature pawn

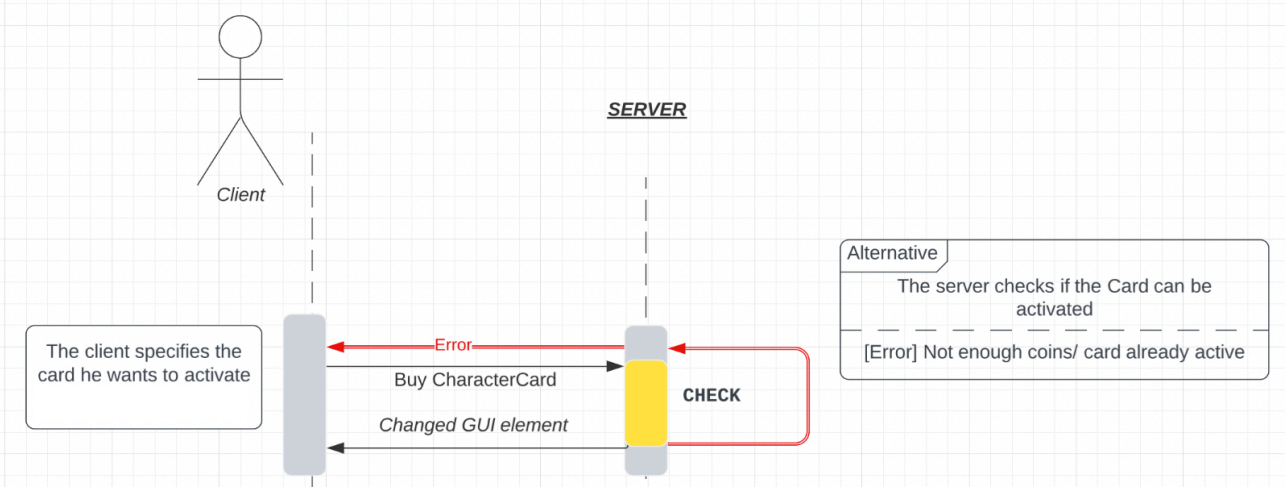
While checking the player is still active through “ping” & “pong” signals, when it’s the right time into the player’s hand the server asks to move MotherNature’s pawn, then the player answers with the number of steps he wants the pawn to make. If the answer is valid the server makes the change visible through the GUI, otherwise it responds with an error message.



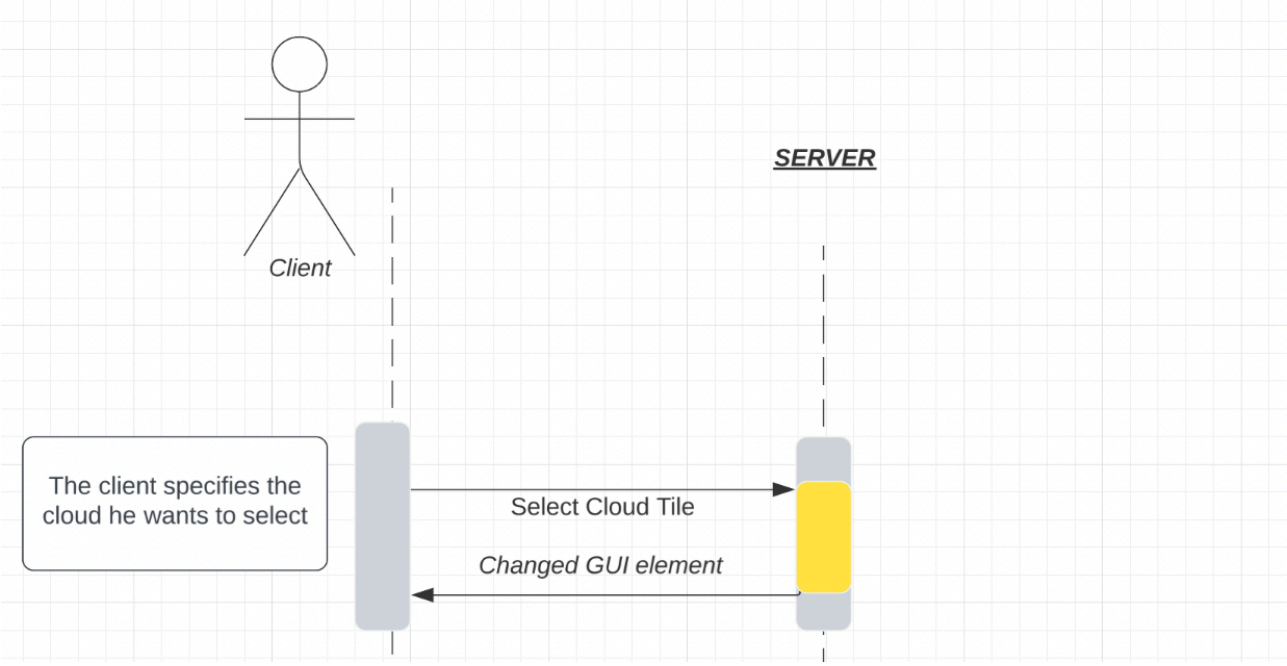
Moving students from the entrance of the SchoolBoard
After checking the player is still active through “ping” & “pong” signals, the Player communicates with the server asking to move a student, the server checks if it’s possible and questions to which location the student needs to be moved to, the player fills up the missing information and the server updates the GUI with all the changes that moving that student entails.



Activating Special Cards
While checking the player is still active through “ping” & “pong” signals, the player can ask the server to buy a Character Card, which then confirms if the option is viable, should that not be the case it sends back an error message explaining the reason why the move was invalid (not enough coins, card is already active). The card may require further interaction depending on the action it offers to the player.



Choosing a CloudTile to end the Client's turn at the game
While checking the player is still active through "ping" & "pong", the players signals the server he wants to choose a CloudTile so the server checks if the choice is valid and applies the changes while also updating the Client's GUI.



- **Deserialization of messages**

To decompose the messages that the server receives we were thinking of using a Visitor Patter that extracts the information necessary to translate the Strings into Java objects. Another option could be the use the GSON library, that allows us to serialize messages and deserialize them thorough methods like **toJson()** and **fromJson().**

pom.xml

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.9</version>
</dependency>
```

Gradle dependency.

build.gradle

```
dependencies {
  implementation 'com.google.code.gson:gson:2.8.9'
}
```