Comunication Protocol

PRE-PARTITA

AGGIUNTA DEI GIOCATORI E AVVIO PARTITA

Una volta avviato il server, esso rimane in attesa di ricevere richieste di connessione dai client.

Il client instaura la connessione con il server, che successivamente richiede all'utente uno username per autenticarsi.

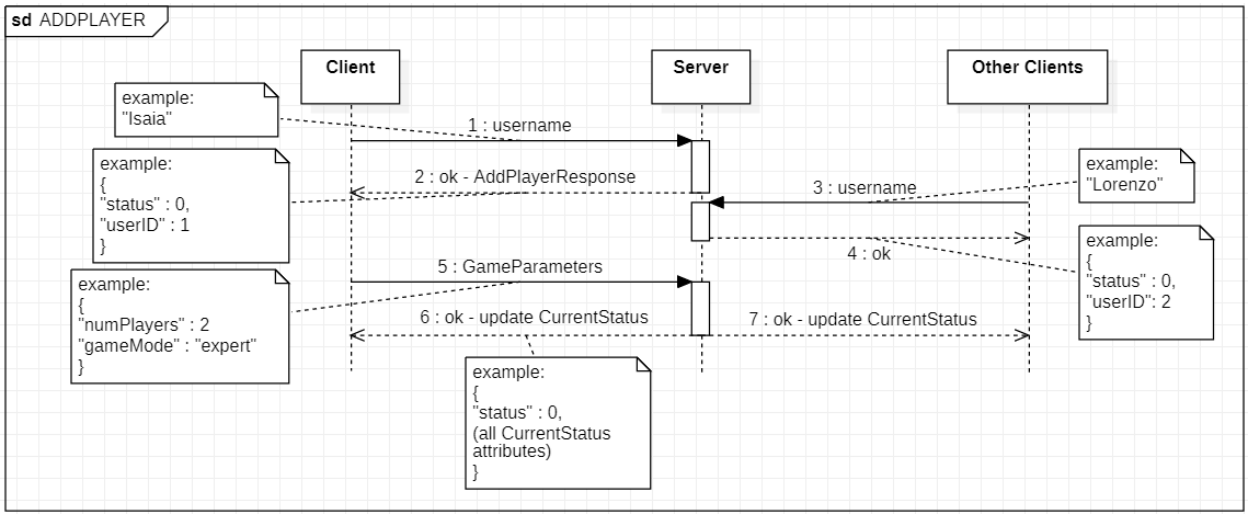
Il client quindi invia al server una stringa contenente unicamente l'username scelto dal giocatore

Il server esegue il parsing e verifica la presenza di eventuali errori; se non presenti, il server risponde con l'userID del giocatore.

Il client, se è il primo giocatore, richiede all'utente la modalità e il numero di giocatori tramite una stringa JSON corrispondente alla seguente classe:

```
public class GameParameters{
  int numPlayers;
  String gameMode;
}
```

Gli altri client connessi attendono unicamente l'avvio della partita.

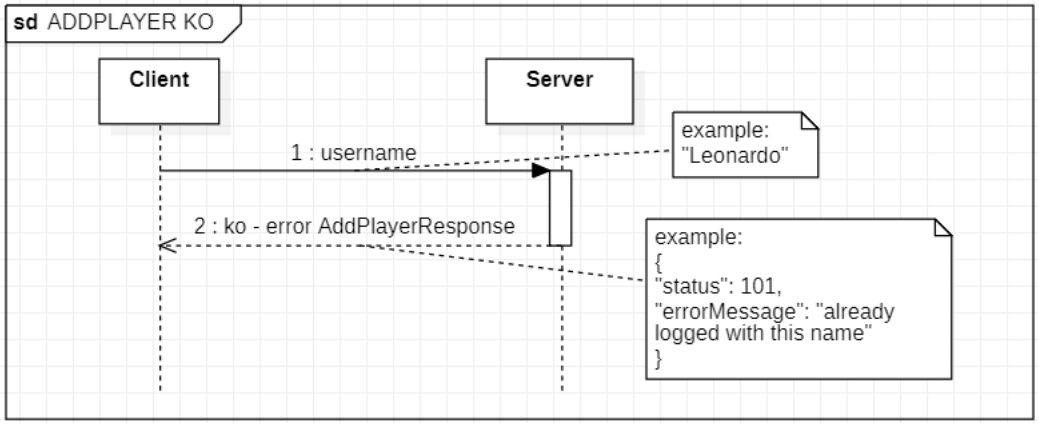


GESTIONE DEGLI ERRORI

Il server, alla ricezione dell'username dell'utente, controlla e gestice la presenza di eventuali errori.

Nel caso di errore, il server risponde con una stringa JSON come segue:

```
{
  "status" : <codice dello stato>,
  "errorMessage" : "<stringa contenete messaggio di errore>";
}
```



Aggiunta di un giocatore con errore.

Il client quindi effettua il parsing del JSON di risposta dal server attraverso la seguente classe:

```
public class AddPlayerResponse {
  int status;
  Integer userID;
  String errorMessage;
}
```

e attraverso le seguenti istruzioni:

```
//jsonString contiene il JSON
Gson g = new Gson();
AddPlayerResponse a = g.fromJson(jsonString, AddPlayerResponse.class);
```

Il messaggio di errore, invece, viene gestito attraverso il seguente enumerativo:

```
public enum StatusCode {
  ALREADYLOGGED(101,"Already logged with this name."),
  FULL_LOBBY(102,"Server has already reached the maximum number of players."),
  GAMESTARTED(103,"Game has already started."),
  KICKEDOUT(104, "Host chose to play a game of 2 players, you were the third."),
  //+ errori in fase di gioco

  private final int statusCode;
  private final String errorMessage;

  StatusCode(int statusCode, String errorMessage) {
    this.statusCode = statusCode;
    this.errorMessage = errorMessage;
  }
}
```

GESTIONE PARTITA

INVIO DEI COMANDI AL SERVER

Il client, quando decide di eseguire un’azione, lo fa inviando al server un file json corrispondente alla seguente classe:

```
public class Command {
    String cmd;
    int playerIndex;
    String studentColour;
    int index;
    int motherNatureShifts;
```

```
    //character card parameters
    int pIndex;
    String pColour;
    int[] pStudentsFrom, pStudentsTo;

    public Command() {}
}
```

I comandi sono elencati in un enumerativo come segue:

```
public enum CommandList {
    MOVETOISLAND(/*Classe comando corrispondente*/),
    MOVETOHALL(/*Classe comando corrispondente*/),
    MOVEMOTHERNATURE(/*Classe comando corrispondente*/),
    PLAYASSISTANTCARD(/*Classe comando corrispondente*/),
    PLAYCHARACTERCARD(/*Classe comando corrispondente*/),
    TAKEFROMCLOUD(/*Classe comando corrispondente*/);

    private final CommandStrategy command;

    CommandList(CommandStrategy cmd) {
      command = cmd;
    }
}
```

In Command, la stringa cmd deve essere una delle stringhe associate agli elementi di CommandList, altrimenti ritorno un messaggio di errore; gli altri attributi sono eventuali parametri associati al comando (es. MOVETOISLAND ha bisogno di playerIndex, ossia il giocatore che ha attivato il comando, e index, che nel caso di questo comando rappresenta l'indice dell'isola su cui si vuole posizionare lo studente).

Un esempio di file JSON generato dal client potrebbe essere:

```
{
  "cmd" : "MOVETOISLAND",
  "playerIndex" : 0,
  "studentColour" : "yellow",
  "index" : 5
}
```

Il server riceve il file JSON, lo parsa in un oggetto di tipo Command e in base all'attributo cmd chiama le funzioni corrispondenti settando i metodi corrispondenti.

Il parsing verra fatto in questo modo:

```
//jsonString contiene il JSON
Gson g = new Gson();
Command c = g.fromJson(jsonString, Command.class);
```

RISPOSTA DEL SERVER

(Nel caso un comando vada a buon fine)

Il client riceve lo stato aggiornato della partita attraverso un file json che contiene una o più delle seguenti informazioni:

- Stato (per il controllo di eventuali errori)
- Messaggio di errore (in caso di errore)
- Username del vincitore (in caso di vittoria)

Stato del turno corrente - TurnStatus:

- ID del giocatore che può effettuare azioni in questo turno
- Fase del turno (pianificazione / azione)

Stato della board e dei player - GameStatus:

- Lista delle isole (con relativo indice, raggruppate in arcipelaghi) e gli studenti su di esse, ed eventualmente torri su di esse.
- Posizione di madre natura
- studenti sulle nuvole
- carte personaggio e eventuali elementi su di esse (monete, studenti, tessere divieto)

Poi: per ogni giocatore

- Carte assistente rimaste
- Carta assistente in cima alla “pila degli scarti” (ossia l’ultima giocata)
- monete
- stato della dashboard (studenti in entrance, hall e torri)

Ogni comando restituisce un JSON contenente solo le informazioni che vengono modificate a causa di esso

Il parsing di GSON, in caso di valori non descritti nella stringa JSON, lascia tali attributi ai valori di default (null in caso di oggetti, valori di default in caso di tipi primitivi, es. 0 per gli int, false per i boolean).

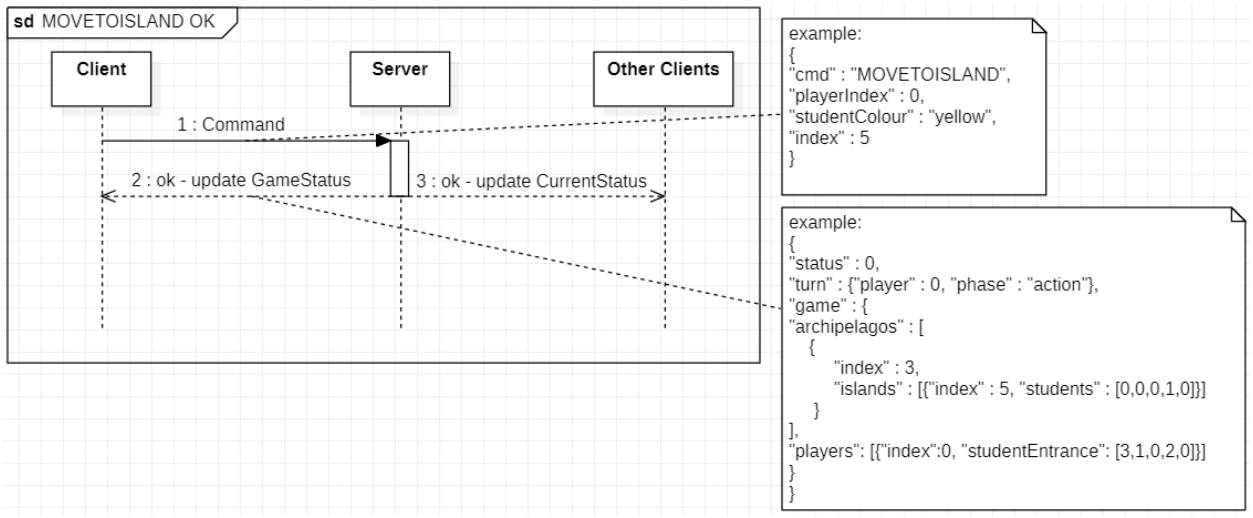
La classe CurrentStatus è la rappresentazione del formato dei dati che verranno inviati al client (Il client quindi parserà il JSON in un oggetto della classe CurrentStatus)

```
public class CurrentStatus{
  int status;
  String errorMessage;
  String winner;
  TurnStatus turn;
  GameStatus game;
}

public class TurnStatus{
  int player;
  String phase;
}

public class GameStatus {
    Integer motherNatureIndex;
    ArchipelagoStatus[] archipelagos;
    CloudStatus[] clouds;
    CharacterCardsStatus[] characterCards;
    PlayerStatus[] players;
}

public class ArchipelagoStatus {
    int index;
    IslandStatus[] islands;
}

public class IslandStatus {
    int islandIndex;
    int[] students;
    String towerColour;
}

public class CloudStatus {
    int index;
    int[] students;
}

public class CharacterCardsStatus {
    int index;
    String fileName;
    Boolean coinOnIt;
    Integer noEntryTiles;
    int[] students;
}

public class PlayerStatus {
    int index;
    int[] assistantCardsLeft;
    Integer lastAssistantCardPlayed;
    Integer coins;
    int[] studentsEntrance;
```

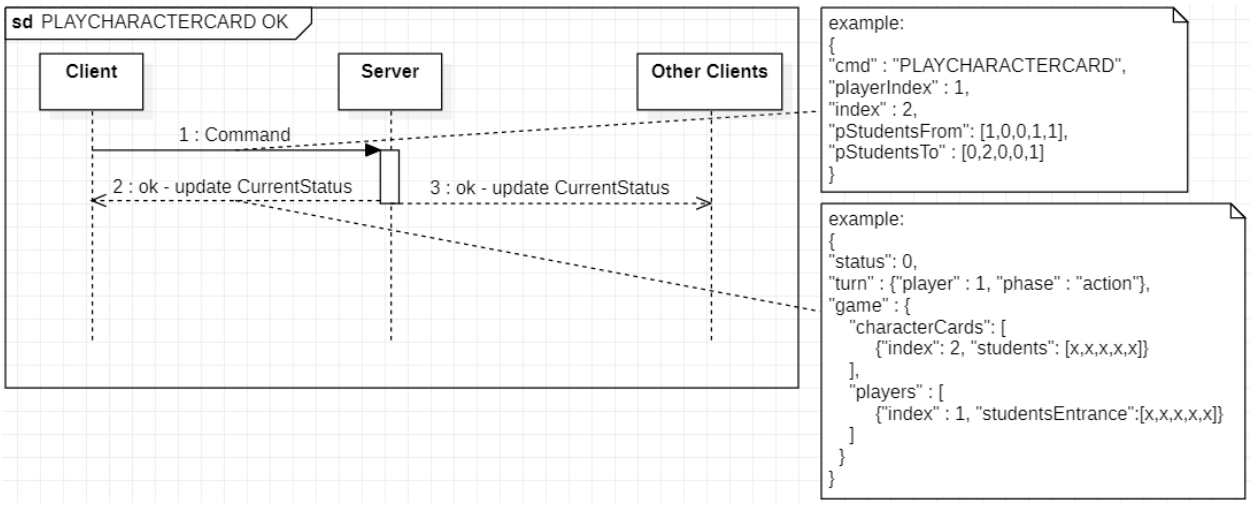
```
    int[] studentsHall;
    Integer numTowers;
}
```

Una versione “completa” di file JSON è quella che viene inviata all’inizio della partita (esempio):

```
{
  "status" : 0,
  "turn" : {"player" : 0, "phase" : "planning"},
  "game" : {
    "motherNatureIndex": 1, //in realtà e' casuale all'inizio
    "archipelagos" : [
      {
        "index": 1,
        "islands" : [{"index": 1, "students": [0,0,0,0,0]}] //nessuna torre quindi non scrivo questa informazione
      },
      {
        "index": 2,
        "islands" : [{"index": 2, "students": [1,0,0,0,0]}]
      },
      {
        "index": 3,
        "islands" : [{"index": 3, "students": [0,1,0,0,0]}]
      },
      {
        "index": 4,
        "islands" : [{"index": 4, "students": [0,0,1,0,0]}]
      },
      ...
      {
        "index": 7,
        "islands" : [{"index": 7, "students": [0,0,0,0,0]}]
      },
      ...
      {
        "index": 12,
        "islands" : [{"index": 12, "students": [0,0,0,0,0]}]
      }
    ],
    "clouds" : [{"index": 0, "students": [x,x,x,x,x]},{"index": 1, "students": [x,x,x,x,x]}],
    "characterCards": [{"index": 0, "fileName": "P02.png"},{"index": 1, "fileName": "P08.png"},{"index": 2, "fileName": "P12.png"}],
    //altre informazioni di characterCards le aggiungo solo se servono
    "players": [
      {"index":0, "assistantCardsLeft": [0,1,2,3,4,5,6,7,8,9], "coins":1, "studentsEntrance":[x,x,x,x,x], "studentsHall":[x,x,x,x,x], "numT
      {"index":1, "assistantCardsLeft": [0,1,2,3,4,5,6,7,8,9], "coins":1, "studentsEntrance":[x,x,x,x,x], "studentsHall":[x,x,x,x,x], "numT
    ]
  }
}
```



command MOVETOISLAND on ok.



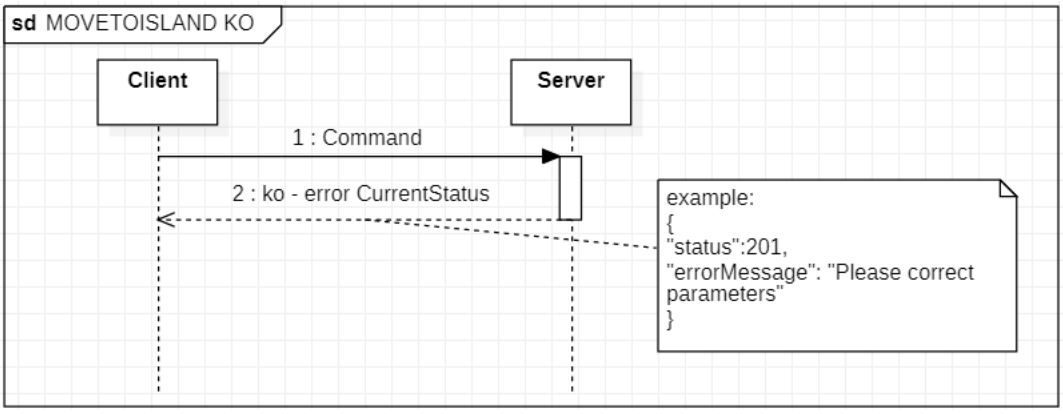
command PLAYCHARACTERCARD on ok.

GESTIONE DEGLI ERRORI

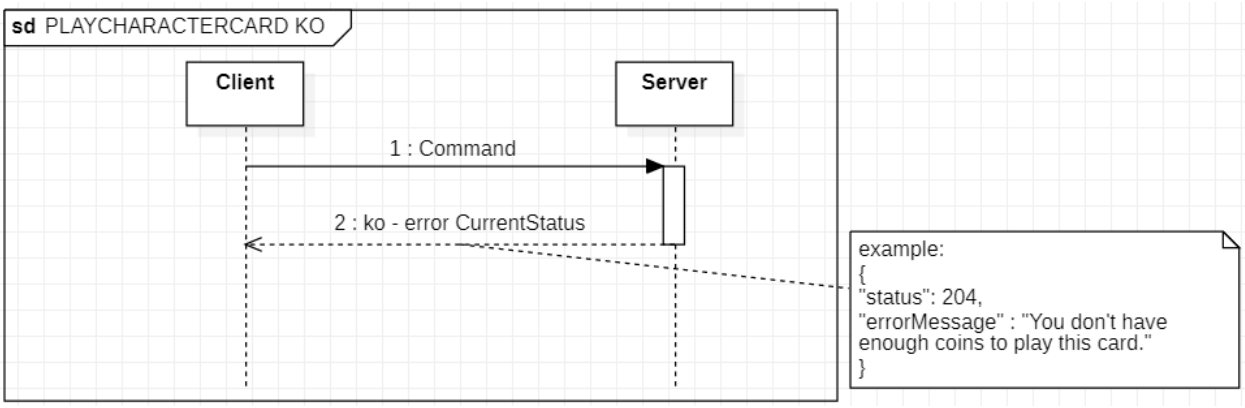
Il server, alla ricezione dell'username dell'utente, controlla e gestice la presenza di eventuali errori.

Nel caso di errore, il server risponde con una stringa JSON in formato GameStatus ma contenente unicamente lo status code e la stringa contenete il messaggio di errore, estratti tra gli elementi del seguente enumerativo:

```
public enum StatusCode {
  //+ errori in fase di pre-partita
  ILLEGALARGUMENT(201,"Please correct parameters"),
  ALREADYPLAYEDAC(202,"You already played this card! Choose another one."),
  ALREADYPLAYEDCC(203,"You already playerd a character card! Wait until next turn"),
  NOTENOUGHCOINS(204, "You don't have enough coins to play this card."),
  FULLHALL(205,"Hall is full."),
  NOSTUDENTS(206,"Not enough students."),
  WRONGTURN(207,"You can only play during your turn"),
  WRONGPHASE(208,"You can't do this move on the current phase of the turn"),

  private final int statusCode;
  private final String errorMessage;

  StatusCode(int statusCode, String errorMessage) {
    this.statusCode = statusCode;
    this.errorMessage = errorMessage;
  }
}
```



command MOVETOISLAND on KO.



command PLAYCHARACTERCARD on KO.

GESTIONE FINE PARTITA

Nel momento in cui il server rileva la vittoria di un giocatore, viene inviata unicamente la stringa winner di CurrentStatus.

Il client mostrerà quindi la relativa schermata.