Comunication Protocol (eng)

PRE-GAME

ADDING PLAYERS AND STARTING THE GAME

Once the server starts, it waits for connection requests from clients.

The client establishes a connection with the server, which subsequently asks the user for a username to authenticate.

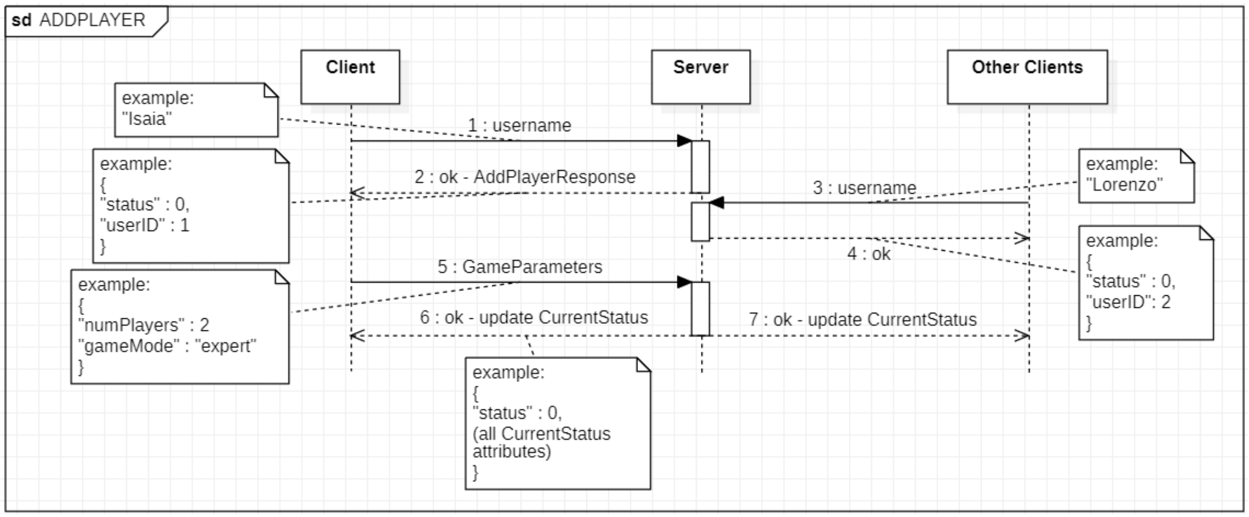
The client then sends the server a string containing only the username chosen by the player.

The server parses and checks for errors; if not present, the server responds with the player's userID.

The client, if it is the first player, asks the user for the mode and number of players via a JSON string corresponding to the following class:

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

The other connected clients are only waiting for the game to start.

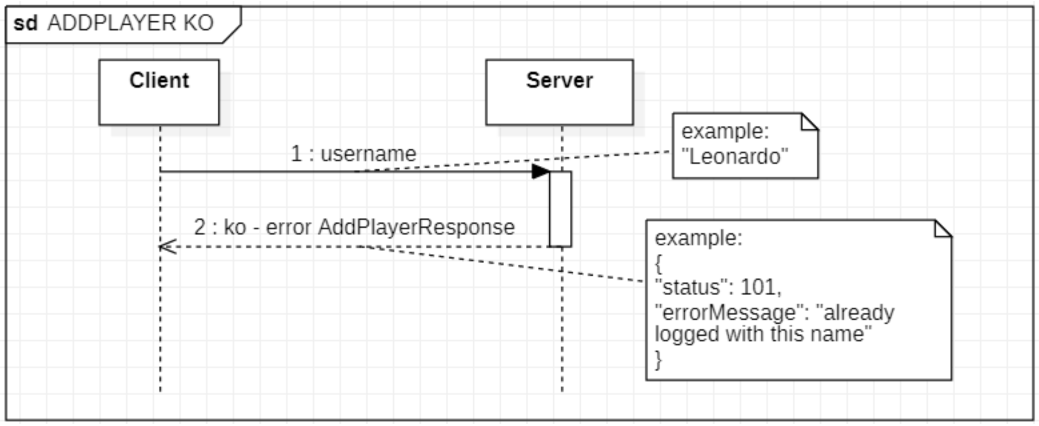

sd ADDPLAYER
Client
Server
Other Clients
example: "Isaia"
1 : username
example: { "status" : 0, "userID" : 1 }
2 : ok - AddPlayerResponse
3 : username
example: "Lorenzo"
4 : ok
example: { "status" : 0, "userID": 2 }
5 : GameParameters
example: { "numPlayers" : 2 "gameMode" : "expert" }
6 : ok - update CurrentStatus
7 : ok - update CurrentStatus
example: { "status" : 0, (all CurrentStatus attributes) }

ERROR MANAGEMENT

Upon receipt of the user's username, the server checks and manages the presence of any errors.

In the event of an error, the server responds with a JSON string as follows:

```
{
  "status" : <status code>,
  "errorMessage" : "<string which contains the error message>";
}
```



Error on adding players.

The client then parses the response JSON from the server through the following class:

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

and through the following instructions:

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

The error message, on the other hand, is handled through the following enum class:

```
public enum AddPlayerStatusCode {
  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.");
  //+ in-game errors

  private final int statusCode;
  private final String errorMessage;

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

GAME MANAGEMENT

SENDING COMMANDS TO THE SERVER

When the client decides to perform an action, it does so by sending a json file corresponding to the following class to the server:

```
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() {}
}
```

The commands are listed in an enumeration class as follows:

```
public enum CommandList {
    MOVETOISLAND(/*Corresponding strategy command class*/),
    MOVETOHALL(/*Corresponding strategy command class*/),
    MOVEMOTHERNATURE(/*Corresponding strategy command class*/),
    PLAYASSISTANTCARD(/*Corresponding strategy command class*/),
    PLAYCHARACTERCARD(/*Corresponding strategy command class*/),
    TAKEFROMCLOUD(/*Corresponding strategy command class*/);

    private final String cmdName;

    CommandList(String cmdName) {
        this.cmdName = cmdName;
    }
}
```

In Command, the cmd string must be one of the strings associated with the CommandList elements, otherwise I return an error message; the other attributes are any parameters associated with the command (e.g. MOVETOISLAND needs playerIndex, i.e. the player who activated the command, and index, which in the case of this command represents the index of the island on which the student is to be positioned).

An example of a client generated JSON file might be:

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

The server receives the JSON file, parses it in a Command type object and based on the cmd attribute calls the corresponding functions by setting the corresponding methods.

The parsing will be done like this:

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

SERVER RESPONSE

(In case a command is successful)

The client receives the updated game status via a json file that contains one or more of the following information:

- Status (to check for any errors)
- Error message (in case of an error)
- Username of the winner (in case of victory)

Current turn status - TurnStatus:

- ID of the player who can send commands this turn
- Phase of the turn (planning / action)

Status of the board and players - GameStatus:

- List of islands (with relative index, grouped into archipelagos) and the students on them, and possibly towers on them.

- Mother Nature position
- Students on clouds
- character cards and any elements on them (coins, students, no entry tiles)

Then: for each player

- Assistant cards left
- Assistant card at the top of the "discard pile" (the last played)
- coins
- dashboard status (students in the entrance, hall and towers)

Each command returns a JSON containing only the information that is changed due to it.

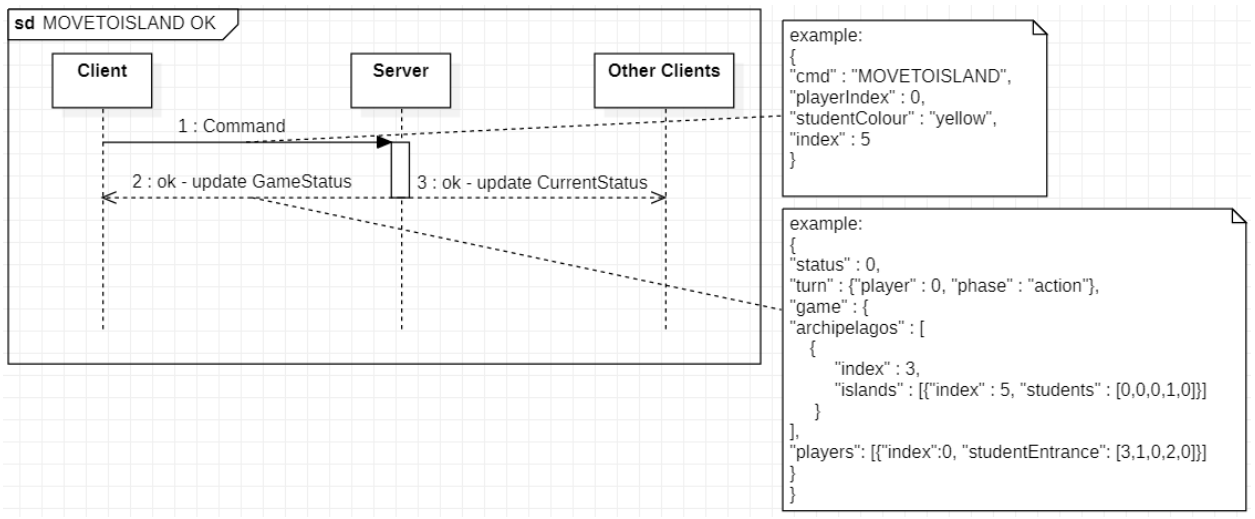
GSON parsing, in case of values not described in the JSON string, leaves these attributes at default values (null in case of objects, default values in case of primitive types, e.g. 0 for int, false for boolean) .

The GameStatus class is the format representation of the data that will be sent to the client (The client will then parse the JSON into an object of the GameStatus class).

```
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;
}
```

A "full" version of a JSON string is the one that is sent at the start of the game:

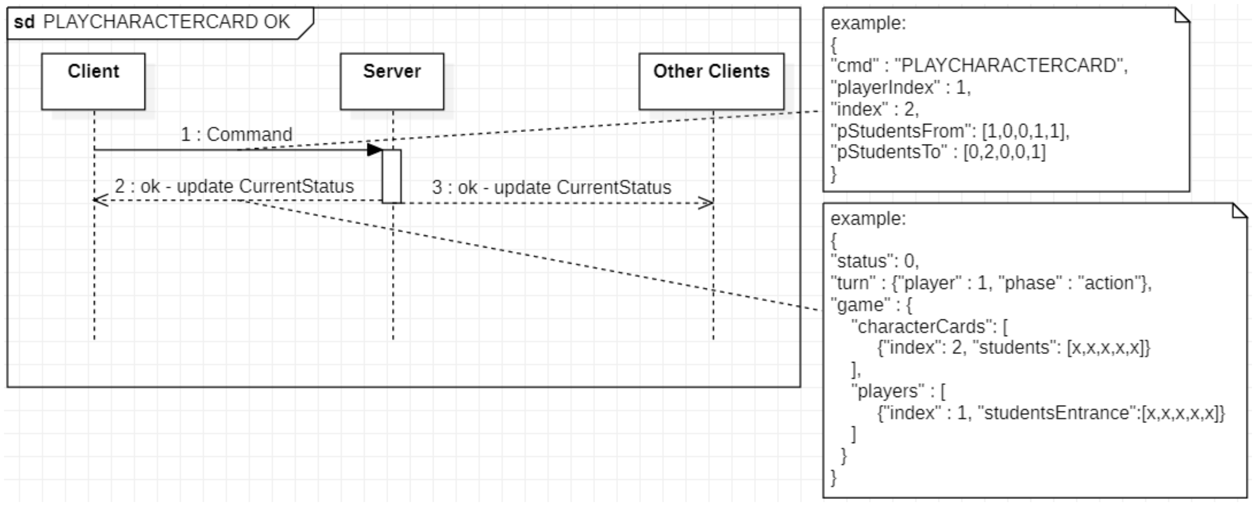
```
{
  "status" : 0,
  "turn" : {"player" : 0, "phase" : "planning"},
  "game" : {
    "motherNatureIndex": 1, //in realtà e' casuale all'inizio
    "archipelagos" : [
      {
        "index": 1,
        "islands" : [{"index": 1, "students": [0,0,0,0,0]}] //nessuna torre quindi non scrivo questa informazione
      },
      {
        "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
    ]
  }
}
```



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

```
example:
{
"status" : 0,
"turn" : {"player" : 0, "phase" : "action"},
"game" : {
"archipelagos" : [
    {
        "index" : 3,
        "islands" : [{"index" : 5, "students" : [0,0,0,1,0]}]
    }
],
"players": [{"index":0, "studentEntrance": [3,1,0,2,0]}]
}
}
```

command MOVETOISLAND on ok.


sd PLAYCHARACTERCARD OK
Client
Server
Other Clients
1 : Command
2 : ok - update CurrentStatus
3 : ok - update CurrentStatus
example:
{
"cmd" : "PLAYCHARACTERCARD",
"playerIndex" : 1,
"index" : 2,
"pStudentsFrom": [1,0,0,1,1],
"pStudentsTo" : [0,2,0,0,1]
}
example:
{
"status": 0,
"turn" : {"player" : 1, "phase" : "action"},
"game" : {
"characterCards": [
{"index": 2, "students": [x,x,x,x,x]}
],
"players" : [
{"index" : 1, "studentsEntrance":[x,x,x,x,x]}
]
}
}

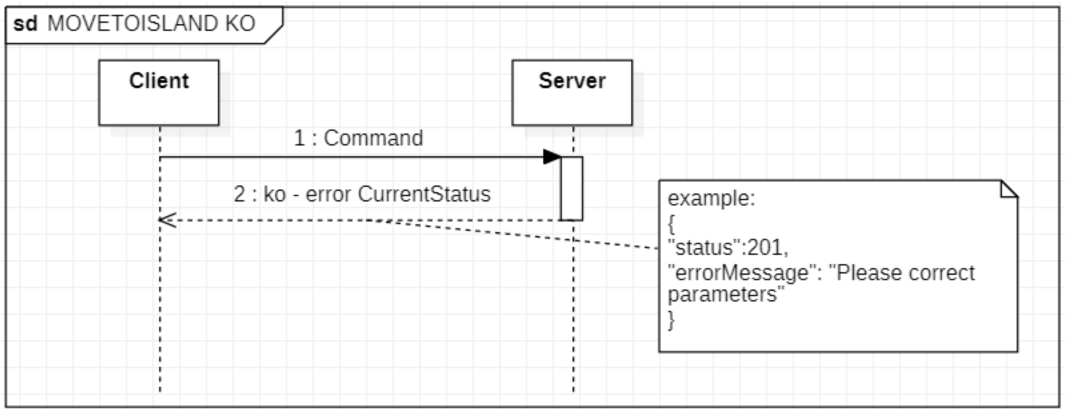
command PLAYCHARACTERCARD on ok.

ERROR MANAGEMENT

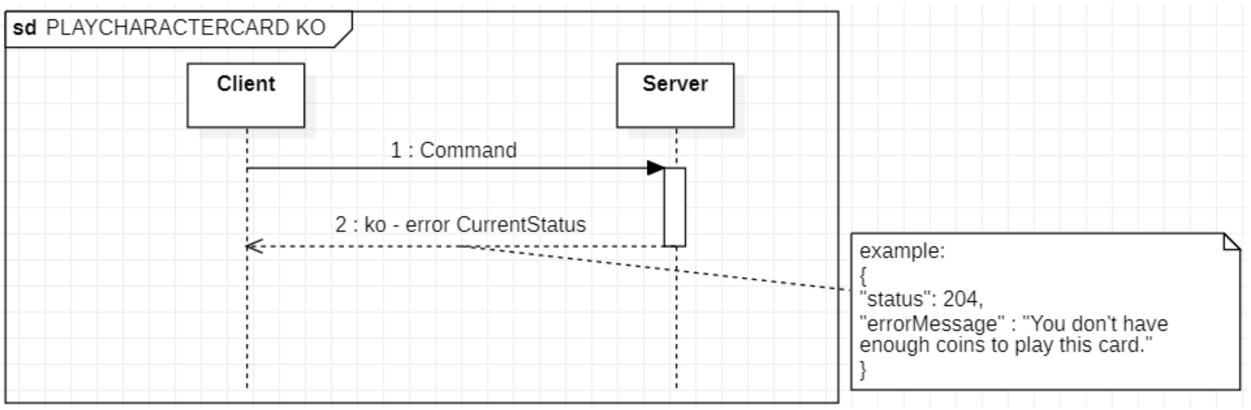
Upon receipt of the user's username, the server checks and manages the presence of any errors.

In the event of an error, the server responds with a JSON string in GameStatus format but containing only the status code and the string containing the error message, extracted from the elements of the following enumeration class:

```
public enum StatusCode {
  //+ pre-game errors
  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;
  }
}
```


sd MOVETOISLAND KO
Client
Server
1 : Command
2 : ko - error CurrentStatus
example:
{
"status":201,
"errorMessage": "Please correct parameters"
}

command MOVETOISLAND on KO.

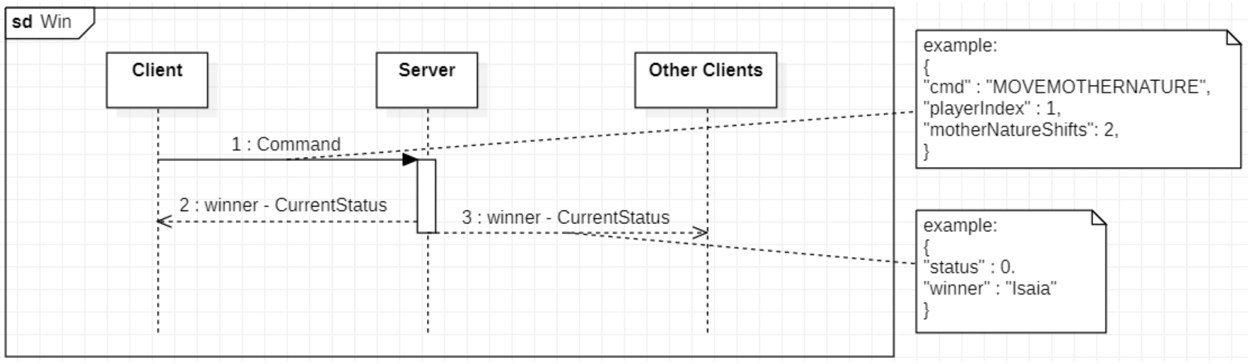

sd PLAYCHARACTERCARD KO
Client
Server
1 : Command
2 : ko - error CurrentStatus
example:
{
"status": 204,
"errorMessage" : "You don't have enough coins to play this card."
}

command PLAYCHARACTERCARD on KO.

END GAME

When the server detects a player's victory, only the CurrentStatus string winner is sent.

The client will then show the winning message.


sd Win
Client
Server
Other Clients
1 : Command
2 : winner - CurrentStatus
3 : winner - CurrentStatus
example:
{
"cmd" : "MOVEMOTHERNATURE",
"playerIndex" : 1,
"motherNatureShifts": 2,
}
example:
{
"status" : 0.
"winner" : "Isaia"
}