Sequence diagrams

<Marco Conti>, <Davide Corradina>, <Flavio De Lellis>, <Nicola De March>
Gruppo <AM26>

We decided to separate the different classes and methods that allow a communication between client and server. Below is a list that contains all the methods that will be discussed next.

All the messages and the data are converted in JSON before sending them.

Client to Server

- Join first player
- Join player
- Leave game
- Pick
- Put
- Order
- Undo
- Chat

Server to Client

- Listener
- Add Player
- Leave game
- Start game
- Player Turn
- Last Round
- End game

JSON format

All the JSON files are made of:

- `Object`: it describes the type of the payload; it could be `message`, `exception`, `listener`.
- `Header`: it contains the command (e.g., `firstJoin`, `put`, `pick`, …).
- `Body`: it contains the data.

Error description

We decide to convert all the exception objects in JSON files which contains the description of each error.

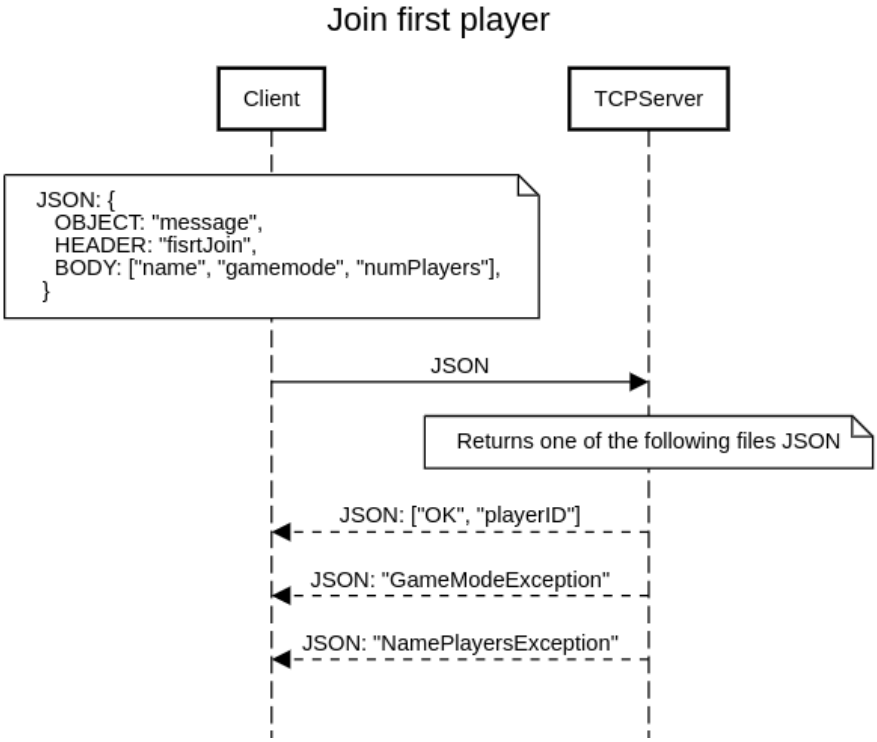
Below an example of the `NameAlreadyExistException`:

```
JSON: {
      Object      : “exception”,
      Header      : “NameAlreadyExistException”,
      Body        : “exception converted to JSON”
}
```

Client to Server

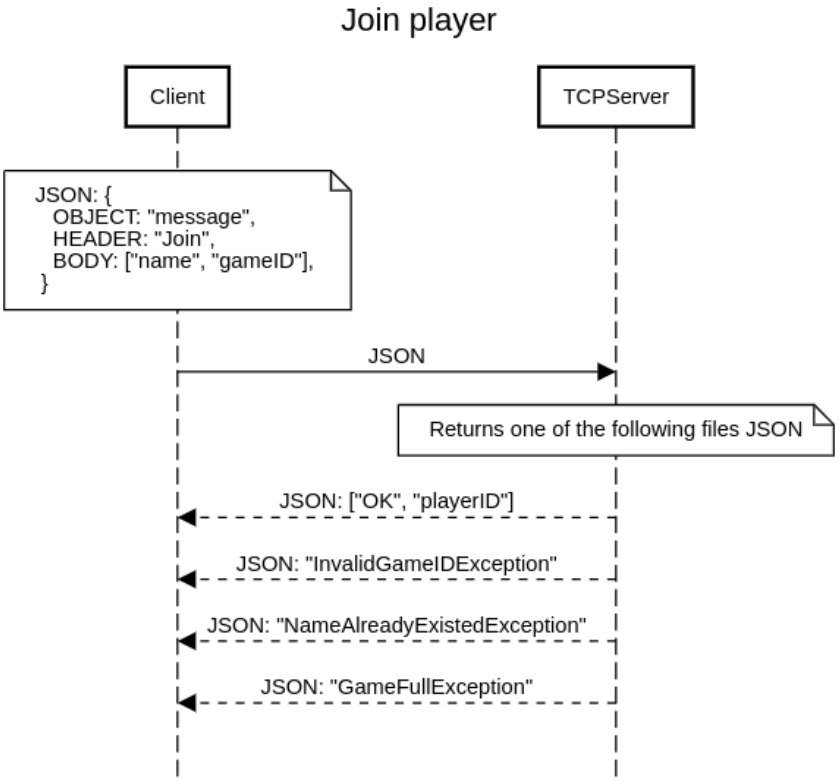
Join first player

This section describes the stage of creating a new game; especially, the create request and response messages between client and server. The message contains the player's name (who creates the game), the game-mode (in our case an enumeration that includes `easy` and `expert` constants) and the maximum number of players who will be able to join the game. On the other hand, the server returns a file JSON with the ID generated for the player if the game creation was successful, otherwise a file JSON with the relative exception.



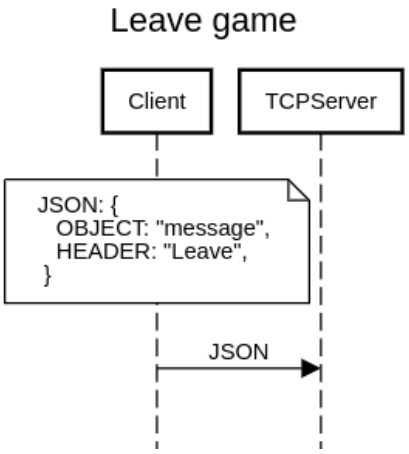
Join

When a player decides to join an existing game, the JSON sent from the client to the server contains the name of the player and the game ID. If the insertion of the player is successful, the server returns the player's ID. If the server fails, returns the relative exception.



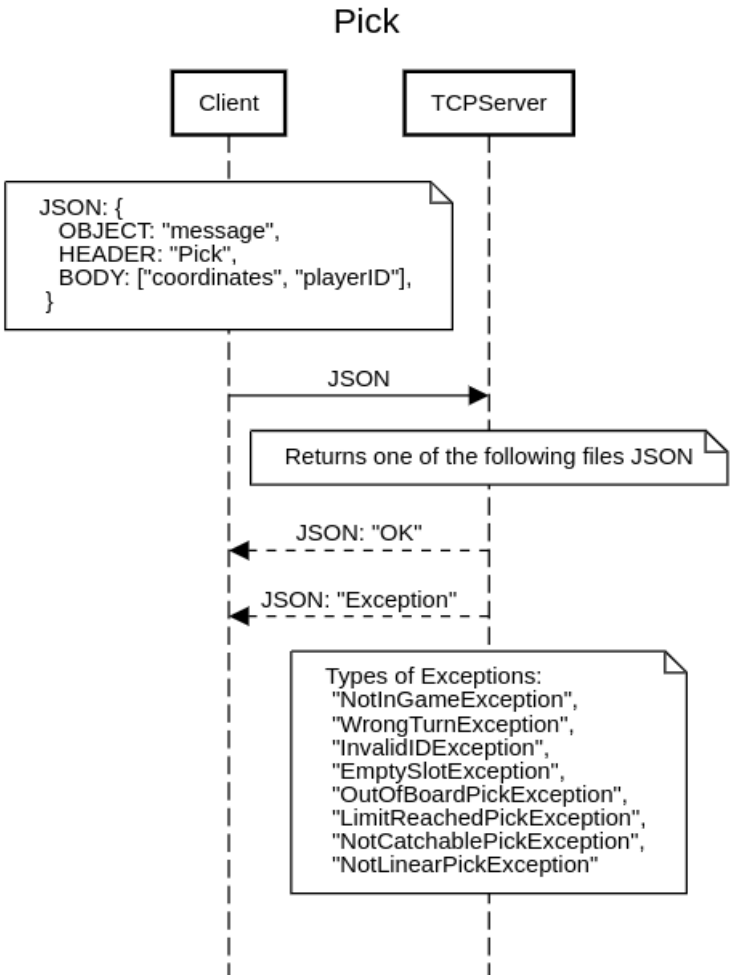
Leave game

If the player wants to quit the game, the client sends a `Leave` message to the server, which closes the player's game without returning any message.



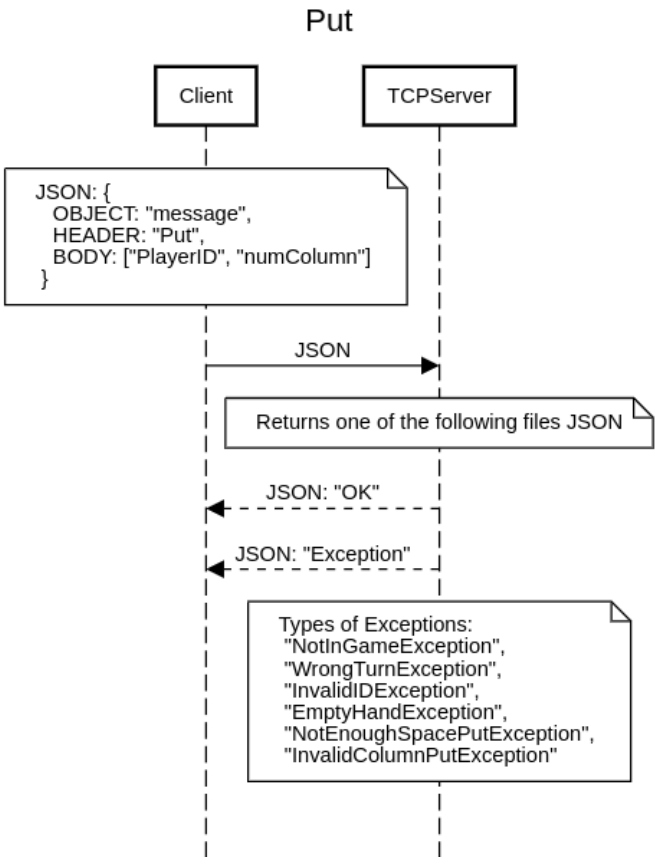
Pick

When a player takes items from the board, the client communicates the coordinates and the player ID. The server returns a JSON that contains a type of exception if the operation fails.



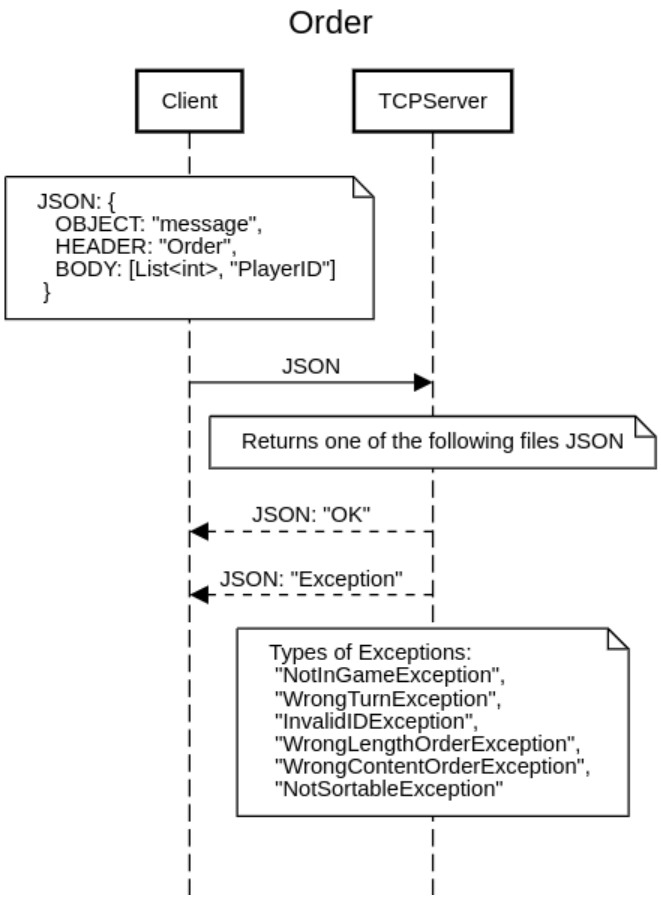
Put

When a player inserts the items, previously selected from the board, the client communicates the number of the column where the player has decided to put the items, in addition to the player ID at issue. The server returns the same messages as in the previous situations.



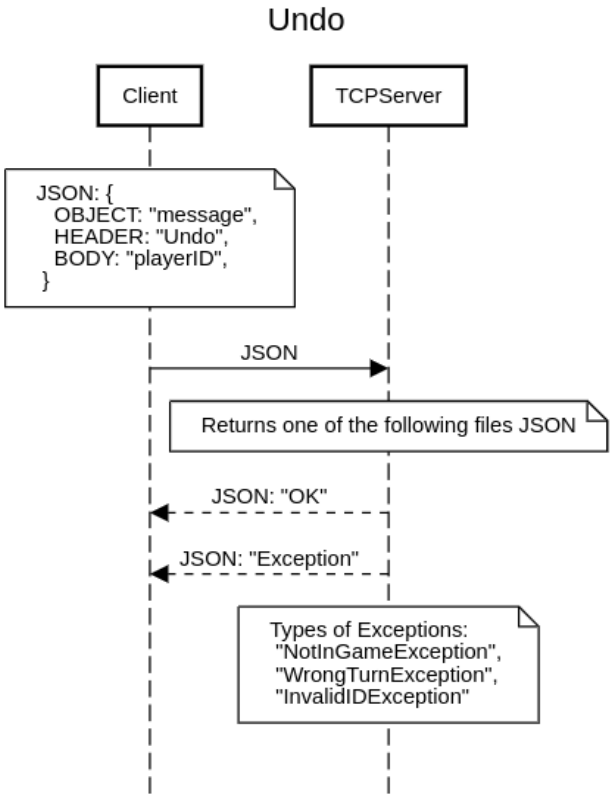
Order

The client communicates the order of insertion of the items in the library. Depending on the number of items selected, the JSON changes. If you have only one Item you don't need to send any order, if you have two then the client will send a message containing two positions, if three the message will have three of them. The server returns the same messages as in the previous situations.

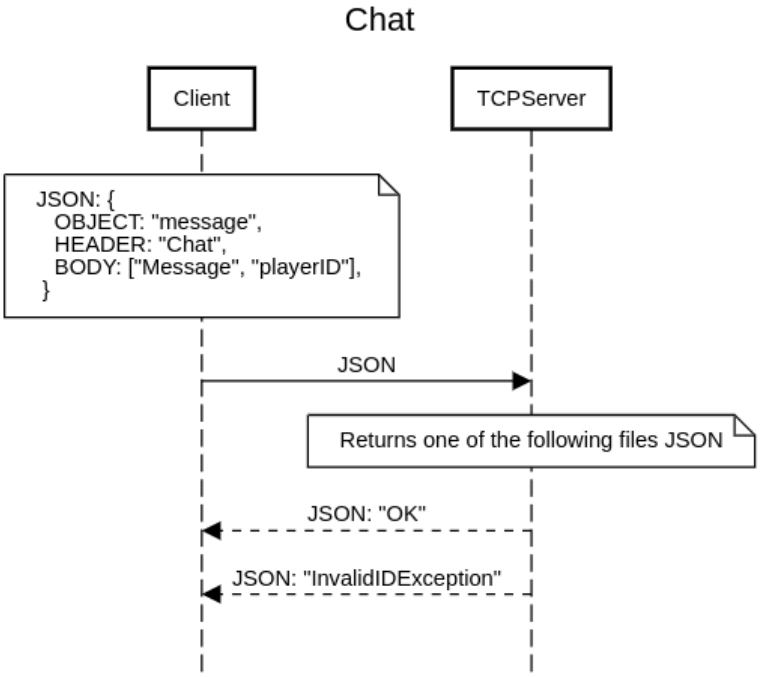


Undo

If the player wants to undo the last action, the client sends an `Undo` message to the server, which returns the same messages as in the previous situations.



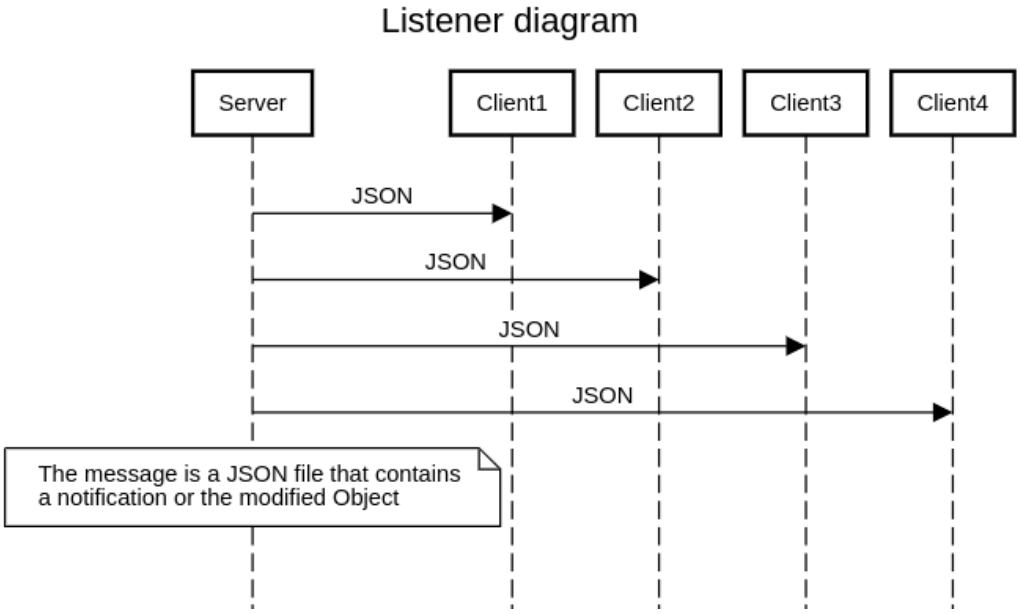
Chat



Server to Client

Listener

When the listener detects a change in the model the server sends a JSON file with the new data to all clients or only to the interested one.



The typical notification of the listener has the following format:

```
JSON: {
      Object      : “listener”,
      Header      : “board”,
      Body        : “data”
}
```

We decided that all the notifications and the messages directed to the clients are managed by the listener. Below is a list of all the other messages that the listener could send to the client during the game.

Add Player

When a player joins a game, the server sends a message to all the other players in the same game with the name of the new player.

```
JSON: {
      Object      : “message”,
      Header      : “newPlayer”,
      Body        : “name”
}
```

Leave

When a player leaves the game, the server communicates to the other players the name of the one who quit the game.

```
JSON: {
      Object      : “message”,
      Header      : “leavePlayer”,
      Body        : “name”
}
```

Player Turn

```
JSON: {
      Object      : “message”,
      Header      : “playerTurn”,
      Body        : “playerName”
}
```

Last Round

```
JSON: {
      Object      : “message”,
      Header      : “lastRound”,
      Body        : “playerName”
}
```

Start Game

When the game starts the server sends a message to all the clients connected with the same `gameID` (a numerical code which identifies the game).

```
JSON: {
      Object      : “message”,
      Header      : “startGame”,
      Body        : “gameID”
}
```

End game

When the game ends, the server notifies all the clients about the game status.

```
JSON: {
      Object      : “message”,
      Header      : “endgame”,
      Body        : “gameID”
}
```