Network Protocol

AM05 - Carrara, Curnis, Demiri, Evi

The following defines the network protocol used to communicate between the server and its clients to notify actions and propagate changes. It starts showcasing the overall syntax designed for general use and then moves on to specific messages, employing sequence diagrams to help the reader visualize game phases. A deep understanding of the game rules, the model and the controller is recommended.

1 Syntax

The syntax was designed to be as simple as possible. Calls to the server implement the following pattern:

```
<method>
{
    "<argument>": <argument>,
    ...
}
```

Listing 1: Method call syntax.

The response can either signal a successful update and return the new model:

```
Game
{
    "game": Game
}
```

Listing 2: Successful update syntax.

Or throw an exception message as a log:

```
Log
{
    "log": String
}
```

Listing 3: Log throw syntax.

Logs are also used for smaller updates, such as chat messages.

2 Sequence Diagrams

What follows applies the syntax above to the specific messages required by the game phases. Each game phase then follows a specific sequence diagram.

2.1 Create Game

2.1.1 joinServer

```
joinServer
{
    "nickname": String
}
```

Listing 4: joinServer call.

Exceptions are thrown if the nickname is null or a player with the same nickname has already joined.

2.2 joinGame

```
joinGame
{
}
```

Listing 5: joinGame call.

Throwable exceptions are `ConnectionRefused` and `FirstConnection`.

2.3 setNumberOfPlayers

```
setNumberOfPlayers
{
    "numberOfPlayers": int
}
```

Listing 6: setNumberOfPlayers call.

Throwable exceptions are `FirstConnection` and `InvalidNumUsers`.

2.3.1 Sequence Diagram

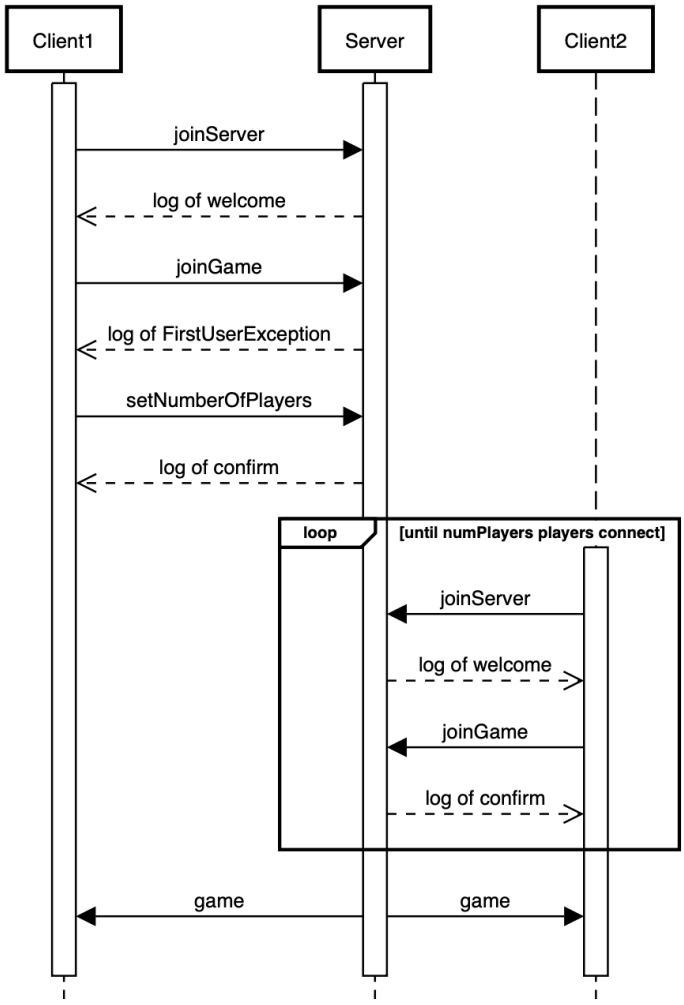

Figure 1: Sequence diagram of the create game phase.

2.4 Place Starter Sides

2.4.1 placeStarterSide

```
placeStarterSide
{
    "isFront": boolean
}
```

Listing 7: placeStarterSide call.

The only throwable exception is `MoveNotAllowed`.

2.4.2 Sequence Diagram

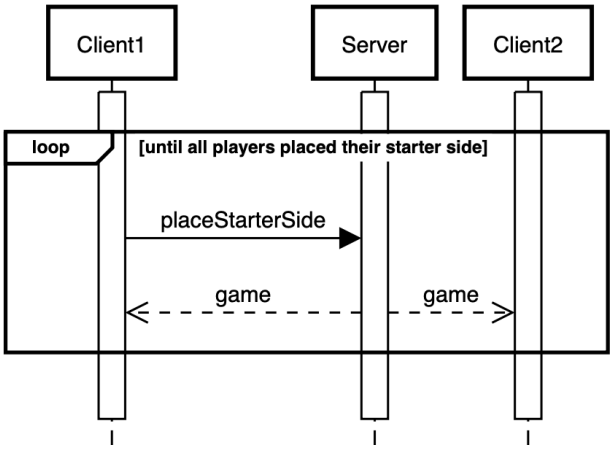

Figure 2: Sequence diagram of the place starter sides phase.

2.5 Choose Objectives

2.5.1 chooseObjective

```
chooseObjective
{
    "objectiveId": String
}
```

Listing 8: chooseObjective call.

Throwable exceptions are `MoveNotAllowed` and `ObjectiveNotAllowed`.

2.5.2 Sequence Diagram

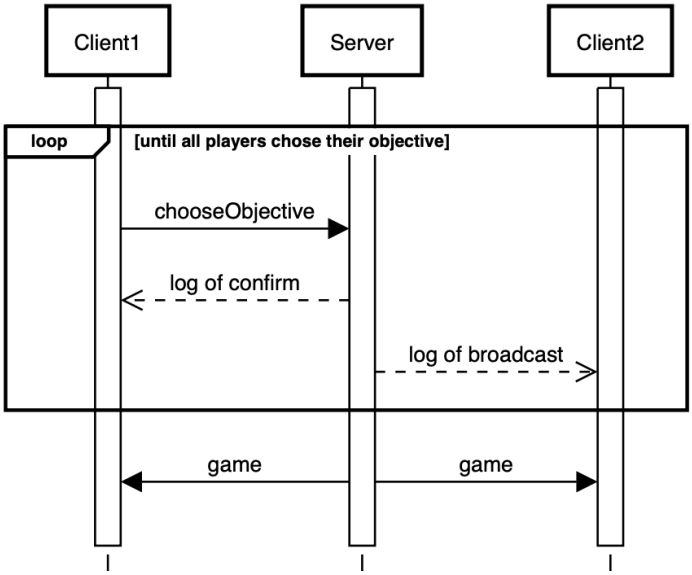

Figure 3: Sequence diagram of the choose objectives phase.

2.6 Place Sides

2.6.1 placeSide

```
placeSide
{
    "cardId": String,
    "isFront": boolean,
    "i": int,
    "j": int
}
```

Listing 9: placeSide call.

Throwable exceptions are:

- `NotYourTurn`;
- `MoveNotAllowed`;
- `InvalidCard`;
- `InvalidCoordinates`;
- `PlacementNotAllowed`;

2.6.2 drawVisible

```
drawVisible
{
    "cardId": String
}
```

Listing 10: drawVisible call.

Throwable exceptions are `NotYourTurn`, `MoveNotAllowed` and `InvalidVisibleCard`.

2.6.3 drawDeck

```
drawDeck
{
    "isGold": boolean
}
```

Listing 11: drawDeck call.

Throwable exceptions are `NotYourTurn`, `MoveNotAllowed` and `EmptyDeck`.

2.6.4 Sequence Diagram

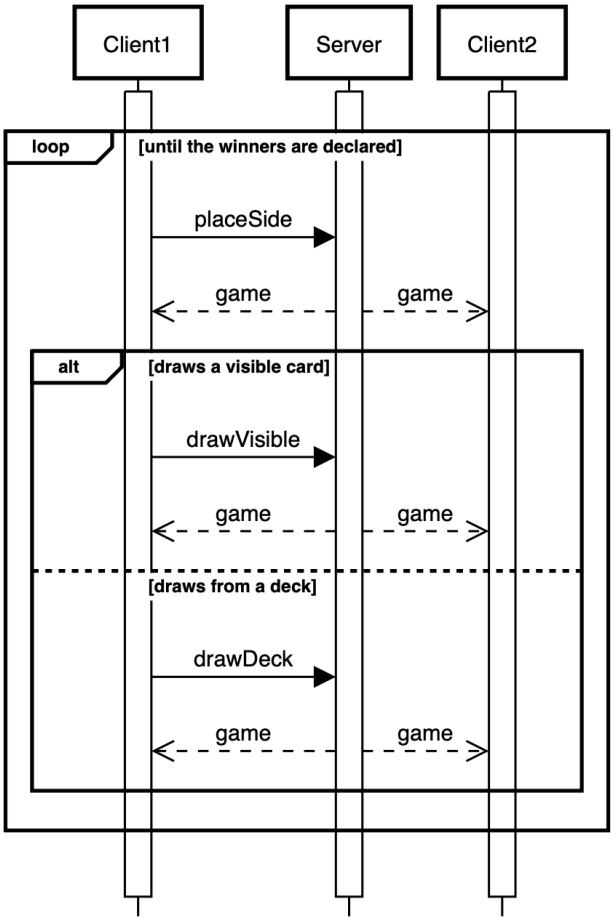

Figure 4: Sequence diagram of the place sides phase.

2.7 Chat

2.7.1 message

```
message
{
    "message": String
}
```

Listing 12: message call.

Exceptions are thrown if the game has not started or the player is not part of the game.

2.7.2 directMessage

```
directMessage
{
    "message": String,
    "recipient": String
}
```

Listing 13: directMessage call.

Exceptions are thrown if the recipient is not part of the game or the recipient is the sender himself.

2.7.3 Sequence Diagram

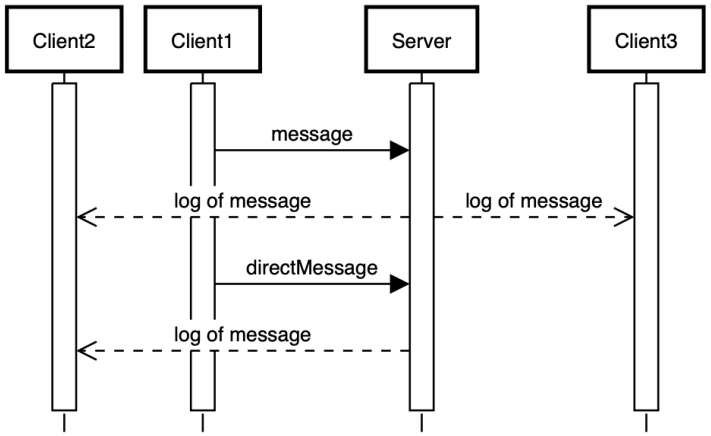

Figure 5: Sequence diagram of the chat.

2.8 Persistence

2.8.1 Sequence Diagram

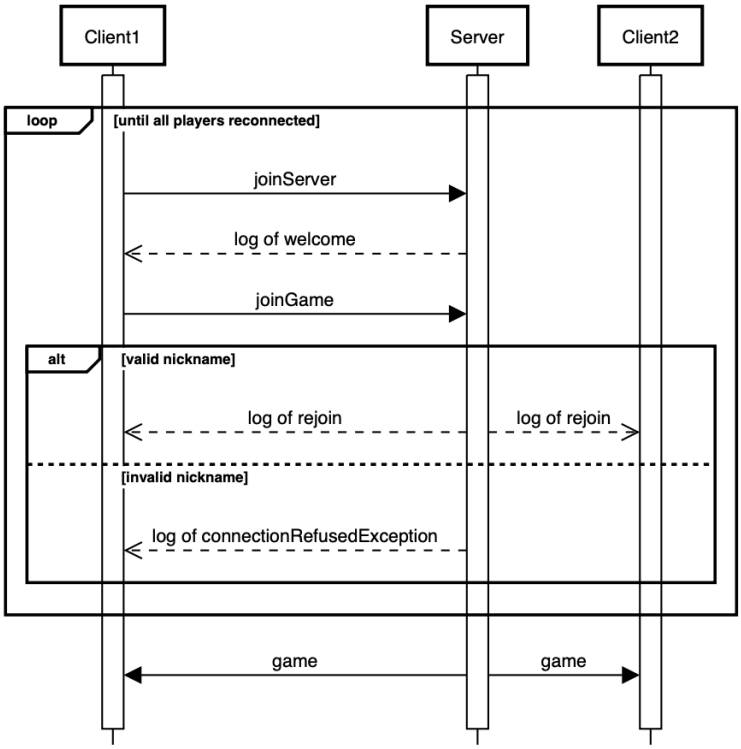

Figure 6: Sequence diagram of persistence.

2.9 Resilience

2.10 disconnect

```
disconnect
{
}
```

Listing 14: disconnect call.

No exceptions are throwable.

2.10.1 Sequence Diagram

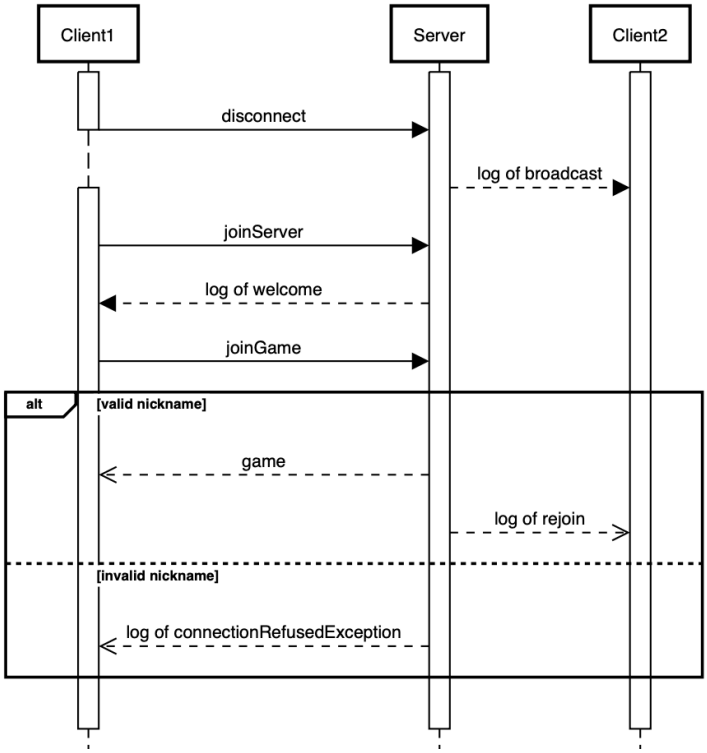

Figure 7: Sequence diagram of the resilience phase.