Communication protocol

AM27

Software Engineering Project
Academic Year 2019/2020

Contents

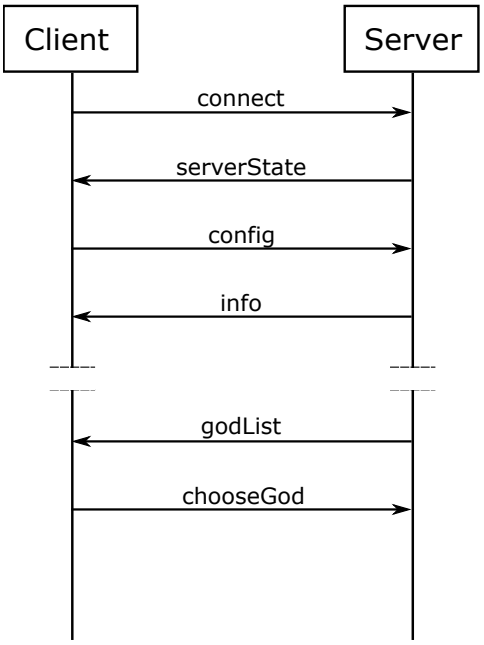

Figure 1: Example of connection and initialization

1 Introduction

This document specifies the standardized communication protocol used in the group's Santorini project. The protocol is based on the JSON format to allow different applications to read and interpret the various messages and commands. Every message after the connection follows the format:

```
{
    "type": "TYPE",
    "message": "{
        ...
    }"
}
```

Where the type identifies the following message, which needs to be deserialized. The second serialization's details to comply with the JSON format (In particular the escaping of " characters) have been omitted to favour readability.

Once the server accepts a client an heartbeat message system is established (using the type `PING`), to assure that every party is still active.

2 Connection

The client connection phase is executed after a client inserts an IP address and port, so the server can be reached. After the ping message the server communicates whether there is a game waiting; consequently the client sends a registration message of type (A) or (B).

2.1 serverState

Message type: Server → Client

```
{
    "type": "SERVERSTATE",
    "message": "{
        "open": true,
        "active": false
    }"
}
```

The message contains two boolean values, specifying if the client has been accepted (`open`) and if there is a match already waiting for players (`active`).

2.2 register (A)

Message type: Client → Server

```
{
    "type": "REGISTER",
    "message": "{
        "nickname": "PLAYER",
        "nPlayers": 2,
        "gods": true
    }"
}
```

This message is sent if there is no active game waiting on the server; contains the configuration for the game.

2.3 register (B)

Message type: Client → Server

```
{
    "type": "REGISTER",
    "message": "{
        "nickname": "PLAYER"
    }"
}
```

This message is sent if there is already an active game waiting on the server.

3 Game initialization

As per the game's rules, in order to start the game the players must choose a god, then the starter; these operations are done through the following messages.

3.1 Choosing a god

3.1.1 godList

Message type: Server → Client

```
{
    "type": "GODLISTMESSAGE",
    "message": "{
         "player": "PLAYER",
         "godList": [
            {
                "name": "GOD",
                "description": "This god..."
            },
            {...}
        ]
    }"
}
```

The message contains the name of the active player and the list of gods that can still be chosen.

3.1.2 chooseGod

Message type: Client → Server

```
{
    "type": "GODINIT",
    "message": "{
        "player": "PLAYER",
        "godName": "GOD"
    }"
}
```

This message contains the god chosen by a player.

3.2 Choosing the starting player

3.2.1 players

Message type: Server → Client

```
{
    "type": "PLAYERSMESSAGE",
    "message": "{
        "player": "PLAYER",
        "players": {
            ["P1", "GOD1"],
            ["P2", "GOD2"],
            ["P3", "GOD3"],
        }
    }"
}
```

This message is sent to everyone to let them know the opponents and their respective gods; the challenger has to choose the starter player from the list.

3.2.2 chooseStarter

Message type: Client → Server

```
{
    "type": "STARTER",
    "message": "{
        "player": "P1",
        "starter": "P2"
    }"
}
```

This is the message sent by the challenger after choosing the starting player.

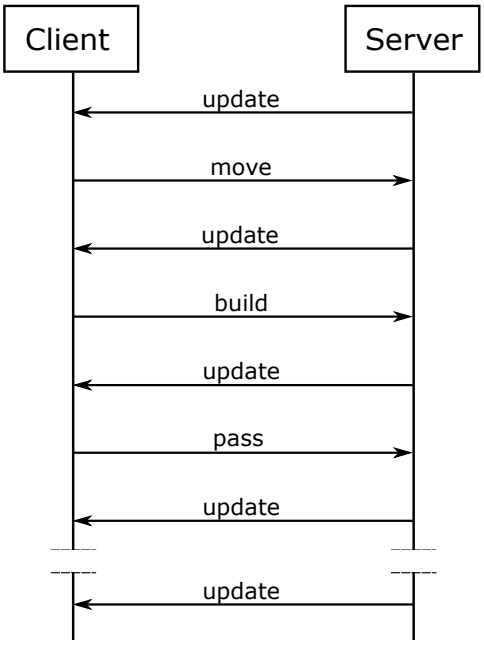

Figure 2: Example of a normal interaction during a turn

4 Game

The following sections illustrate the messages exchanged during a game.

4.1 Server updates

The updates sent by the server regulate entirely the clients' behaviour.

4.1.1 update

```
{
    "player": "PLAYER",
    "boardUpdate": [
        [
            { "row": 1, "column": 1 },
            {
                "levelList": [ 1, 2 ],
                "workerOwner": "PLAYER1",
                "workerSex": "M"
            }
        ],
```

```
        [...]
    ],
    "reachableCells": [
        [
            { "row": 1, "column": 1 },
            [
                { "row": 1, "column": 2 },
                [...]
            ]
        ],
        [...]
    ],
    "buildablecells": [
        [
            { "row": 1, "column": 1 },
            [
                [
                    { "row": 1, "column": 2 },
                    [ 1, 2 ]
                ],
                [...]
            ],
            [...]
        ],
        [...]
    ],
    "canPlaceWorker": false,
    "canPass": false,
    "canUndo": true
}
```

This message contains all the information needed to update a client:

- `player`: Specifies the nickname of the active player.
- `boardUpdate`: Maps all the coordinates which changed during the update to the heights of their buildings and to the (optional) worker on top.
- `reachableCells`: Lists for each worker the cells it can reach.
- `buildableCells`: Lists for each worker all the cells on which it can build, to which are mapped the possible heights.

- `canPlaceWorker`: Boolean that specifies if the client can place a worker on the board.
- `canPass`: Boolean variable set true if the player can pass the turn.
- `canUndo`: Boolean value specifying if the player can undo its last action.

This client-specific message is sent to every player anytime a modification occurs.

4.1.2 info

```
{
    "type": "INFO",
    "message": {
        "player": "RECIPIENT",
        "information": "MESSAGECODE"
    }
}
```

Message used to send a specific player a code representing a string message to be displayed.

4.1.3 endGame

```
{
    "type": "ENDGAME",
    "message": {
        "player": "PLAYER",
        "isWinner": true
    }
}
```

This message notifies the clients that the game terminated; if the `isWinner` field is empty then the game terminated due to the disconnection of the specified player.

4.2 Client commands

The list of commands a client can send, depending by server updates.

4.2.1 workerInit

```
{
    "type": "WORKERINIT",
    "message": {
        "player": "PLAYER",
        "coordinates": { "row": 1, "column": 1 }
        "sex": 'M'
    }
}
```

The workerInit message is used to tell the server where to put a worker.

4.2.2 move

```
{
    "type": "MOVE",
    "message": {
        "player": "PLAYER",
        "source": { "row": 1, "column": 1 },
        "destination": { "row": 2, "column": 2 }
    }
}
```

The message contains information about the movement of a worker, giving its current and new coordinates.

4.2.3 build

```
{
    "type": "BUILD",
    "message": {
        "player": "PLAYER",
        "source": { "row": 1, "column": 1 },
        "destination": { "row": 2, "column": 2 },
        "level": 1
    }
}
```

This message requests a new build to the server.

4.2.4 undo

```
{
    "type": "UNDO",
    "message": {
        "player": "PLAYER",
        "isAll": false
    }
}
```

This message requests the undo of the player's last action. The `isAll` parameter allows the player to choose if undo a single action or the whole turn.

4.2.5 pass

```
{
    "type": "PASS",
    "message": {
        "player": "PLAYER"
    }
}
```

This is the command sent at the end of a player's turn, releasing the control.

4.2.6 lose

```
{
    "type": "LOSE",
    "message": {
        "player": "PLAYER"
    }
}
```

This command is sent to the server when the client determines that the player cannot perform any move, removing them from the game.