netorcai Documentation

Release 2.0.0

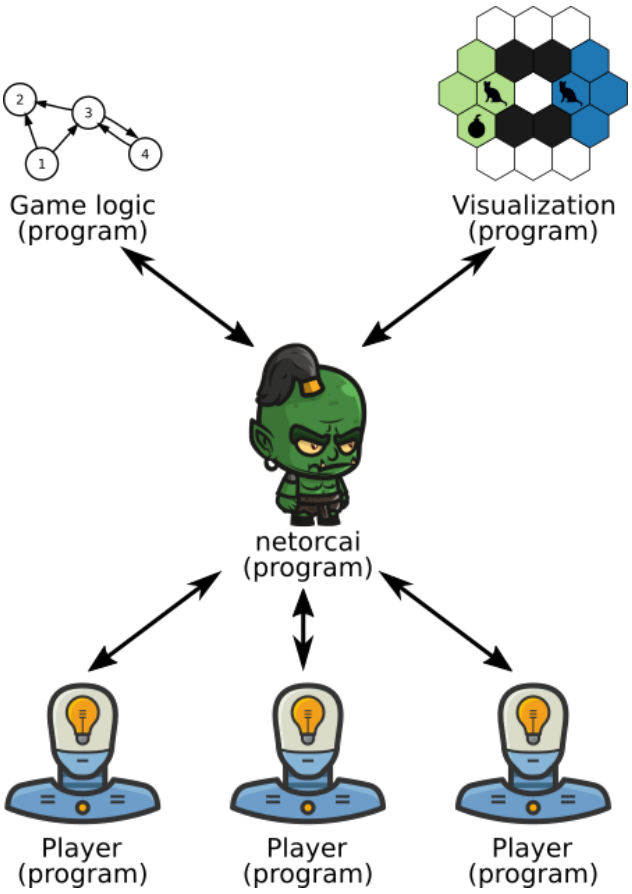
Millian Poquet

Feb 23, 2019

Contents:

netorcai (/net..ka/) is a network orchestrator for artificial intelligence games. It splits a classical game server process into two processes, allowing to develop various games in any language without having to manage all network-related issues about the clients.

This is done thanks to a *Network metaprotocol*.



CHAPTER 1

Installation

1.1 Via go standard tools

As netorcai is implemented in Go, it can be built with the go command. Installation steps are as follows.

1. Install a recent Go version.
2. Run `go get github.com/netorcai/netorcai/cmd/netorcai`. This will download and compile netorcai. The executable will be put into `${GOPATH}/bin` (if the `GOPATH` environment variable is unset, it should default to `${HOME}/go` or `%USERPROFILE%\go`).

In brief.

```
go get github.com/netorcai/netorcai/cmd/netorcai
${GOPATH:-${HOME}/go}/bin/netorcai --help
```

1.2 Via Nix

Nix is a package manager with amazing properties that is available on Linux-like systems. It stores all the packages in a dedicated directory (usually `/nix/store`), which avoids interfering with classical system packages (usually in `/usr`).

Once Nix is installed on your machine (instructions on Nix's web page), packages can be installed with `nix-env --install` (`-i`). The following command shows how to install netorcai with Nix.

```
# Install latest release.
nix-env -f https://github.com/netorcai/netorcaipkgs/archive/master.tar.gz -iA netorcai

# Alternatively, install latest commit.
nix-env -f https://github.com/netorcai/netorcaipkgs/archive/master.tar.gz -iA␣
↪netorcai_dev
```

CHAPTER 2

Network metaprotocol

This metaprotocol is based on TCP and is *mostly* textual, as all messages are composed of two parts.

1. *CONTENT_SIZE*, a 32-bit little-endian unsigned integer corresponding to the size of the message content (therefore excluding the 4 octets used to store CONTENT_SIZE). *CONTENT_SIZE* value must be smaller than 1 kio for the first message, and smaller than 16 Mio for other messages.
2. *CONTENT*, an UTF-8 string of CONTENT_SIZE octets, terminated by an UTF-8 *Line Feed* character (U+000A).

The content of each message must be a valid JSON object. Messages are typed (see *message types*) and clients must follow a specified behavior (see *expected client behavior*).

2.1 Network entities (endpoints)

This metaprotocol allows multiple entities to communicate.

- The unique **game logic** entity, in charge of managing the game itself.
- **Clients** entities, that are in one of the following types.
  - *Player*, in charge of taking actions to play the game
  - *Visualization*, in charge of displaying the game progress
- The unique **netorcai** entity: Central orchestrator (broker) between the game logic and the clients.

Todo: Make a non-ugly entities figure.

2.2 Message types

Each message has a type. This type is set as a string in the `message_type` field of the main message JSON object. The other fields of the main JSON object depend on the message type.

List of messages between **clients** and **netorcai**.

- *LOGIN*
- *LOGIN_ACK*
- *KICK*
- *GAME_STARTS*
- *GAME_ENDS*
- *TURN*
- *TURN_ACK*

List of messages between **netorcai** and **game logic**.

- (*LOGIN*)
- (*LOGIN_ACK*)
- (*KICK*)
- *DO_INIT*
- *DO_INIT_ACK*
- *DO_TURN*
- *DO_TURN_ACK*

2.2.1 LOGIN

This message type is sent from (**clients** or **game logic**) to **netorcai**.

This is the first message sent by clients and game logic. It allows them to indicate they want to participate in the game. **netorcai** answers this message with a *LOGIN_ACK* message if the logging in is accepted, or by a *KICK* message otherwise.

Fields.

- `nickname` (string): The name the clients wants to have. Must respect the `\A\S{1,10}\z` (in go regular expression syntax).
- `role` (string). Must be `player`, `visualization` or `game logic`.
- `metaprotocol_version` (string). The netorcai metaprotocol version used by the client (see *Changelog*).

Example.

```
{
  "message_type": "LOGIN",
  "nickname": "strutser",
  "role": "player",
  "metaprotocol_version": "2.0.0"
}
```

2.2.2 LOGIN_ACK

This message type is sent from **netorcai** to (**clients** or **game logic**).

It tells a client or the game logic that its *LOGIN* has been accepted.

Fields.

- `metaprotocol_version` (string). The netorcai metaprotocol version used by the netorcai program (see *Changelog*).

Example.

```
{
  "message_type": "LOGIN_ACK",
  "metaprotocol_version": "2.0.0"
}
```

2.2.3 KICK

This message type is sent from **netorcai** to (**clients** or **game logic**).

It tells a client (or game logic) that it is about to be kicked out of a game. After sending this message, **netorcai** will no longer read incoming messages from the kicked client (or game logic). It also means that **netorcai** is about to close the socket.

It can be sent for multiple reasons:

- As a negative acknowledge to a *LOGIN* message
- If a message is invalid.
  - Its content is not valid JSON.
  - A field is missing or has an invalid value.
  - If a client does not follow its expected behavior (see *expected client behavior*).
- If **netorcai** is about to terminate.

Fields:

- `kick_reason` (string): The reason why the client (or game logic) has been kicked

Example:

```
{
  "message_type": "KICK",
  "kick_reason": "Invalid message: Content is not valid JSON"
}
```

2.2.4 GAME_STARTS

This message type is sent from **netorcai** to **clients**.

It tells the client that the game is about to start.

Fields.

- `player_id`: (integral non-negative number or -1):
  - If the client role is `player`, this is the player's unique identifier.

  - It the client role is `visualization`, this is -1.
- `players_info`: (array of objects): If this message is sent to a `player`, this array is empty. If this message is sent to a `visualization`, this array contains information about each player.
  - `player_id` (integral non-negative number): The unique player identifier.
  - `nickname` (string): The player nickname.
  - `remote_address` (string): The player network remote address.
  - `is_connected` (bool): Whether the player is currently connected to **netorcai**.
- `nb_players` (integral positive number): The number of players of the game.
- `nb_special_players` (integral positive number): The number of special players of the game.
- `nb_turns_max` (integral positive number): The maximum number of turns of the game.
- `milliseconds_before_first_turn` (non-negative number): The number of milliseconds before the first game *TURN*.
- `milliseconds_between_turns` (non-negative number): The minimum number of milliseconds between two consecutive game *TURN*.
- `initial_game_state` (object): Game-dependent content.

Example.

```
{
  "message_type": "GAME_STARTS",
  "player_id": -1,
  "players_info": [
    {
      "player_id": 0,
      "nickname": "jugador",
      "remote_address": "127.0.0.1:59840",
      "is_connected": true
    }
  ],
  "nb_players": 4,
  "nb_special_players": 0,
  "nb_turns_max": 100,
  "milliseconds_before_first_turn": 1000,
  "milliseconds_between_turns": 1000,
  "initial_game_state": {}
}
```

2.2.5 GAME_ENDS

This message type is sent from **netorcai** to **clients**.

It tells the client that the game is finished. The client can safely close the socket after receiving this message.

Fields.

- `winner_player_id` (integral non-negative number or -1): The unique identifier of the player that won the game. Can be -1 if there is no winner.
- `game_state` (object): Game-dependent content.

Example.

```
{
  "message_type": "GAME_ENDS",
  "winner_player_id": 0,
  "game_state": {}
}
```

2.2.6 TURN

This message type is sent from **netorcai** to **clients**.

It tells the client a new turn has started.

Fields.

- `turn_number` (non-negative integral number): The number of the current turn.
- `game_state` (object): Game-dependent content that directly corresponds to the `game_state` field of a *DO_TURN_ACK* message.
- `players_info`: (array of objects): If this message is sent to a `player`, this array is empty. If this message is sent to a `visualization`, this array contains information about each player.
  - `player_id` (integral non-negative number): The unique player identifier.
  - `nickname` (string): The player nickname.
  - `remote_address` (string): The player network remote address.
  - `is_connected` (bool): Whether the player is currently connected to **netorcai**.

Example.

```
{
  "message_type": "TURN",
  "turn_number": 0,
  "game_state": {},
  "players_info": [
    {
      "player_id": 0,
      "nickname": "jugador",
      "remote_address": "127.0.0.1:59840",
      "is_connected": true
    }
  ]
}
```

2.2.7 TURN_ACK

This message type is sent from **clients** to **netorcai**.

It tells netorcai that the client has managed a turn. For players, it contains the actions the player wants to do.

Fields.

- `turn_number` (non-negative integral number): The number of the turn that the client has managed. Value must match the `turn_number` of the latest *TURN* received by the client.
- `actions` (array): Game-dependent content. Must be empty for visualizations.

Example.

```
{
  "message_type": "TURN_ACK",
  "turn_number": 0,
  "actions": []
}
```

2.2.8 DO_INIT

This message type is sent from **netorcai** to **game logic**.

This message initiates the sequence to start the game. **netorcai** gives information to the game logic, such that the game logic can generate the game initial state.

Fields.

- `nb_players` (integral positive number): The number of players in the game.
- `nb_special_players` (integral positive number): The number of special players in the game.
- `nb_turns_max` (integral positive number): The maximum number of turns of the game.

Example.

```
{
  "message_type": "DO_INIT",
  "nb_players": 4,
  "nb_special_players": 0,
  "nb_turns_max": 100
}
```

2.2.9 DO_INIT_ACK

This message is sent from **game logic** to **netorcai**.

It means that the game logic has finished its initialization. It sends initial information about the game, which is forwarded to the clients.

Fields.

- `initial_game_state` (object): The initial game state, as it should be transmitted to clients. Only the `all_clients` key of this object is currently implemented, which means the associated game-dependent object will be transmitted to all the clients (players and visualizations).

Example.

```
{
  "initial_game_state": {
    "all_clients": {}
  }
}
```

2.2.10 DO_TURN

This message type is sent from **netorcai** to **game logic**.

It tells the game logic to do a new turn.

Fields.

- `player_actions` (array): The actions decided by the players. There is at most one array element per player. This array contains objects that must contain the following fields.
  - `player_id` (non-negative integral number): The unique identifier of the player who decided the actions.
  - `turn_number` (non-negative integral number): The turn whose the actions comes from (received from *TURN_ACK*).
  - `actions` (array): The actions of the player. Game-dependent content (received from *TURN_ACK*).

Example.

```
{
  "message_type": "DO_TURN",
  "player_actions": [
    {
      "player_id": 0,
      "turn_number": 0,
      "actions": []
    }
  ]
}
```

2.2.11 DO_TURN_ACK

This message type is sent from **game logic** to **netorcai**.

Game logic has computed a new turn and transmits its results.

Fields.

- `winner_player_id` (non-negative integral number or -1): The unique identifier of the player currently winning the game. Can be -1 if there is no current winner.
- `game_state` (object): The current game state, as it should be transmitted to clients. Only the `all_clients` key of this object is currently implemented, which means the associated game-dependent object will be transmitted to all the clients (players and visualizations).

Example.

```
{
  "message_type": "DO_TURN_ACK",
  "winner_player_id": 0,
  "game_state": {
    "all_clients": {}
  }
}
```

2.3 Expected client behavior

netorcai manages the clients by associating them with a state. In a given state, a client can only receive and send certain types of messages. A client that sends an unexpected type of message is kicked by **netorcai** (see *KICK*).

The following figure summarizes the expected behavior of a client.

- Each node is a client state.

- Edges are transitions between states.
  - ?MSG_TYPE means that the client receives a message of type MSG_TYPE.
  - !MSG_TYPE means that the client sends a message of type MSG_TYPE.

Todo: Make a non-ugly client behavior figure.

2.4 Expected game logic behavior

Similarly to clients, **netorcai** manages the game logic by associating it with a state.

Its expected behavior is described in the following figure.

Todo: Make a non-ugly logic behavior figure.

CHAPTER 3

Client libraries

The netorcai architecture is a client-server one. The netorcai program has the role of a network server while the other entities (games, players and visualizations) have a client role.

While netorcai clients can be implemented from scratch, several libraries have been implemented to ease the communication with the netorcai server. All these libraries are available in the netorcai organization github repository. Currently, the following libraries have been implemented.

- netorcai-client-cpp
- netorcai-client-d
- netorcai-client-fortran
- netorcai-client-java
- netorcai-client-python

Contrary to bindings, all these libraries are fully implemented in the target programming language. The main advantage is that the installation of each library is simplified, as it can be done directly with the language packaging tools.

3.1 Client libraries API

All the client libraries propose the same programming interface. Inner details may of course vary depending on the programming language, such as the type used to store collections of items or the variable/function name depending on the language coding style. All existing libraries provide the following.

- A high-level `Client` class that manages the network connection.
- Structured types for the various messages of the metaprotocol (see *Message types*). Each message is implemented as a `struct` in C++ and D, and as `class` in Java and Python.
- Functions to parse the various metaprotocol messages.

The `Client` class is intended to be the main way to send and receive netorcai messages. This class provides the following methods.

- Various methods to send metaprotocol messages on the network, named `send<MESSAGE_TYPE>` (*e.g.*, `sendLogin`).
- Various methods to receive and parse metaprotocol messages from the network, named `read<MESSAGE_TYPE>` (*e.g.*, `readLoginAck`). **These functions do not return until a message could be read** (or if a connection issue has been detected).
- `sendString` and `sendJson`, that respectively send a user-defined string or a user-defined JSON object on the network.
- `recvString` and `recvJson`, that respectively receive a string or a JSON object from the network. **These functions do not return until a message could be read** (or if a connection issue has been detected).

Note: All these methods can throw exceptions if a network error has been encountered. Furthermore, all `read<MESSAGE_TYPE>` methods will throw an exception if an unexpected message type has been received (*e.g.*, if the client received a *KICK*).

3.2 Usage examples

As an example, here is a basic player bot in Python.

```
try:
    # Instantiate a client in memory.
    client = Client()

    # Connect the internal socket to netorcai (on the 4242 port of the local machine).
    client.connect("localhost", 4242)

    # Log in to netorcai as a player. The client's nickname is "Example".
    client.send_login("Example", "player")
    client.read_login_ack()

    # Wait for the game to start.
    game_starts = client.read_game_starts()

    # Precalculation can be done here. Here, the initial game state is just printed.
    print(game_starts.initial_game_state)

    # For each turn.
    for i in range(game_starts.nb_turns_max):
        # Wait for the turn to start.
        turn = client.read_turn()
        # Decide what to do. Here, the current game state is just printed and no␣
↪action is done.
        print(turn.game_state)
        actions = []
        # Send the decided actions to netorcai.
        client.send_turn_ack(turn.turn_number, [])
except Exception as e:
    print(e)
```

All libraries have examples in the `examples` directory of their respective repository. Please refer to them for more examples.

3.3 Getting the libraries

Getting the latest released version is easy for languages that have a standard package index.

- D: Add the `netorcai-client` dependency in your project (netorcai-client package on DUB).
- Java: Not uploaded on the maven repository yet .
- Python: `pip install netorcai` (netorcai package on PyPI)

Otherwise, getting the library from its git repository is pretty straightforward. Building and installation instructions are in the README of each repository.

Alternatively, some of these libraries are packaged in Nix in the netorcaipkgs package repository. Here are some commands to install the libraries.

```
# Install the C++ client library.
# Latest release
nix-env -f https://github.com/netorcai/netorcaipkgs/archive/master.tar.gz -iA␣
↪netorcai_client_cpp
# Up-to-date (latest commit)
nix-env -f https://github.com/netorcai/netorcaipkgs/archive/master.tar.gz -iA␣
↪netorcai_client_cpp_dev
```

CHAPTER 4

Frequently asked questions

4.1 Running netorcai in my scripts gives an ioctl error

Try using the `--simple-prompt` option.

4.2 Running netorcai in background does not work in my scripts

Try launching netorcai via nohup.

CHAPTER 5

Rationale

In the context of Lionel Martin's challenge, I have been involved in the implementation of multiagent network games meant to be played by bots.

After implementing several games (spaceships in 2014, aquar.iom in 2016) I came to the following conclusions.

- Implenting the network server is tough.
- Handling the clients correctly (errors, fairness, not spamming slow clients…) mostly means that most of the development time is in the network game server, not in the game itself.
- The games in this context are quite specific (fair, turn-based, visualizable, no big performance constraint), which means the development effort can be shared regarding the network server.

CHAPTER 6

Changelog

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog. netorcai adheres to Semantic Versioning and its public API includes the following.

- netorcai's program command-line interface.
- netorcai's metaprotocol.

6.1 Unreleased

- Commits since v2.0.0

6.2 v2.0.0

- Commits since v1.2.0
- Release date: 2019-02-24

6.2.1 Changed (breaks metaprotocol)

- The *CONTENT_SIZE* message header is now 32-bit (was 16-bit). *CONTENT_SIZE* value must be smaller than 1 Kio for the first message, and smaller than 16 Mio for other messages.
- Protocol version handshake added in *LOGIN* and *LOGIN_ACK*. As a result, old clients will not be able to log in anymore because their metaprotocol version is unknown.

6.2.2 Added

- New CLI command `--fast`, which allows to start a new turn as soon as all players have decided what to do — instead of relying on a timer. This assumes that all player clients return in finite time — either by returning a *TURN_ACK* or by closing their sockets.
- Special players can now be connected to netorcai. The game logic knows which bots are *special*, which allows implementing game modes with asymetric game rules. As an example, the ghosts of a bomberman game could be implemented in a special player bot which could have different actions than bombermen.
  - New CLI command `--nb-splayers-max`, to define the maximum number of special players.
  - *DO_INIT* and *GAME_STARTS* messages now contain a `nb_special_players` field.

6.2.3 Fixed

- Various corner case deadlocks have been fixed and should now be covered by integration tests.
- Several issues around netorcai's termination have been fixed.
  - Kicking clients induced a non-compressible time delay to limit the loss of messages. This time delay has been removed **and** the last messages sent by netorcai should **not** be lost anymore.
  - Data races could occur in the sending of the last messages to clients.

6.3 v1.2.0

- Commits since v1.1.0

6.3.1 Added

- New CLI command `--autostart`, that automatically starts the game when all clients (and one game logic) are connected. The expected clients are those defined by `--nb-players-max` and `--nb-visus-max`.

6.3.2 Changed

- Client libraries are now hosted on netorcai's organization github repository.
- Documentation is now on netorcai's readthedocs.

6.3.3 Fixed

- All players always remained connected in the `players_info` array of *GAME_STARTS* and *TURN* messages. Now, the `is_connected` field of disconnected players should be set to `false`.

6.4 v1.1.0

- Commits since v1.0.1
- Release date: 2018-10-29

6.4.1 Added

- New CLI command `--simple-prompt`, that forces the use of the basic prompt.

6.5 v1.0.1

- Commits since v1.0.0
- Release date: 2018-10-23

6.5.1 Changed

- The repository has moved to https://github.com/netorcai/netorcai.

6.6 v1.0.0

- Commits since v0.1.0
- Release date: 2018-06-11

6.6.1 Added (program):

- The metaprotocol is now fully implemented. netorcai is now heavily tested under continuous integration, all coverable code should now be covered.
- New `--delay-turns` command-line option to specify the minimum number of milliseconds between two consecutive turns.
- New interactive prompt.

6.6.2 Changed (metaprotocol):

- *GAME_STARTS*
  - The `data` field has been renamed `initial_game_state`.
  - `player_id`: The "null" player_id is now represented as -1 (was JSON's `null`).
  - New `milliseconds_between_turns` field (minimum amount of milliseconds between two consecutive turns).
  - New `players_info` array used to forward information about the players to visualization clients.

- *GAME_ENDS*
  - The `data` field has been renamed `game_state`.
  - `winner_player_id`: The "null" player_id is now represented as -1 (was JSON's `null`).
- *TURN*
  - New `players_info` array used to forward information about the players to visualization clients.
- *DO_TURN_ACK*
  - New `winner_player_id` field, which represents the current leader of the game (if any).
- The `DO_FIRST_TURN` message type has been renamed *DO_INIT*
- New *DO_INIT_ACK* message (game logic initialization).

6.6.3 Fixed:

- Various fixes, as the metaprotocol was not implemented yet — and therefore not tested.

6.7 v0.1.0

- First released version.
- Release date: 2018-05-02

Todo: Make a non-ugly entities figure.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/netorcai/checkouts/latest/docs/metaprotocol.rst, line 36.)

Todo: Make a non-ugly client behavior figure.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/netorcai/checkouts/latest/docs/metaprotocol.rst, line 457.)

Todo: Make a non-ugly logic behavior figure.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/netorcai/checkouts/latest/docs/metaprotocol.rst, line 470.)