CLI Transcendance

Or how to make things more difficult
than they already are

PalsFrenier (tdelage)

October 7, 2025

Summary

1 Introduction **2**

2 User manual **3**
- 2.1 Build process . . . 3
- 2.2 Configuration . . . 4

3 Subject requirements **5**
- 3.1 CLI Application . . . 5
- 3.2 API Communication . . . 6
- 3.3 Authentication . . . 7
- 3.4 Synchronization . . . 8
- 3.5 Cross Platform . . . 9
- 3.6 Documentation . . . 10

4 Implementation **11**
- 4.1 Code Architecture . . . 11
  - 4.1.1 App structure . . . 11
  - 4.1.2 Main loop implementation . . . 12
  - 4.1.3 Concurrency model . . . 12
- 4.2 Used Libraries . . . 12
  - 4.2.1 Integration notes . . . 13

5 Glossary **14**

Conclusion **16**

1 Introduction

The CLI module of the Transcendance project is a major module of the Server-Side Pong.
The subject introduces it as follows:

> • **Major module**: Replace Basic Pong with Server-Side Pong and Implementing an API.
>
> *In this major module, the goal is to replace the basic Pong game with a server-side Pong game, accompanied by the implementation of an API.*

This paper will provide a complete explanation of the CLI application's use and how I implemented all the features needed to finish the module.

2 User manual

2.1 Build process

Overview The CLI module provides three equivalent ways to compile the application. Each of these methods ultimately relies on the same internal build script, written in C and located at the root of the `cli/` directory. This script, named `build.c`, automates the entire compilation process, including dependency retrieval, environment setup, and binary generation.

Dependencies management Unlike traditional manual builds, the `build.c` script takes care of preparing the build environment automatically. It downloads, builds, and links all required dependencies, such as:

- the **C3 compiler** (`c3c`),
- **libCurl** for HTTPS and WSS requests,
- and **libTermbox** for terminal rendering.

This script is implemented using the `nobuild.h` header from the streamer *Tsoding*, which allows C-based build scripts to replace shell-based build systems. An additional header, `bob.h`, extends this functionality with utility wrappers such as `rm(char*)`, `clone(char*)`, and `make(char*)`, simplifying repetitive operations.

Build methods There are three supported ways to build the project:

1. Using the global project Makefile:

   ```
   $ make cli
   ```

   This command triggers the CLI build process from the root of the Transcendance repository.

2. Using the local Makefile within the `cli/` directory:

   ```
   $ make
   ```

   This method is functionally equivalent to the previous one, but scoped to the CLI submodule only.

3. Compiling and executing the `build.c` file directly:

   ```
   $ gcc build.c
   $ ./build build
   ```

This approach provides full control over the build process and can be useful for debugging or understanding how dependencies are managed internally.

Both Makefiles simply act as frontends to the same build script, invoking `build.c` automatically. Therefore, all three methods are synchronized and produce identical build outputs.

Available build rules The build system defines three main rules, available in both the Makefiles and the internal build script:

- **build** — compiles the entire CLI program and its dependencies;
- **clean** — removes all generated files, including build artifacts and the resulting binary;
- **re** — sequentially executes the `clean` rule followed by the `build` rule, ensuring a full rebuild from scratch.

These rules are available only on the last methods as they are defined to be used with the build script

Build output The compilation generates a single executable named `transcli`, located at the root of the `cli/` directory. This binary can be launched directly from the terminal once the build process completes successfully.

2.2 Configuration

Overview The CLI module requires minimal configuration before execution. The main purpose of the configuration is to define the server address with which the CLI will communicate. All configuration parameters are set at compile time and cannot be changed after compilation.

CLI Configuration File The configuration is handled via a single file, `cli.conf`, automatically generated by the global Makefile. The corrector or user does not modify `gen.c3` directly. The build script, `build.c`, reads `cli.conf` and generates `gen.c3` accordingly.

An example of the configuration file is:

```
addr:="12.25.124.123"
port:="8443"
```

Here, `addr` represents the IP address of the server, and `port` specifies the port to use. The port number (8443) should generally not be changed, as it corresponds to the port used by the Transcendance server.

Immutability Once the CLI program is compiled, the server address is fixed in `gen.c3`. No runtime modifications are possible. Any changes to the server IP require updating `cli.conf` and recompiling the program via the build script.

3 Subject requirements

3.1 CLI Application

Requirements The CLI module is the primary interface for users to interact with the server-side Pong application. Its main requirements are:

- **Server Communication:** The CLI must communicate with the Transcendance server using secure protocols (HTTPS and WSS) to ensure data integrity and real-time synchronization.
- **Authentication:** It must support user authentication, allowing only authorized access to the server functionalities.
- **Cross-Platform Terminal Support:** The CLI should operate correctly in standard Linux terminals and render the game and messages properly.
- **Minimal User Configuration:** Users should not need to modify internal files; configuration is limited to specifying the server address via a centralized configuration file (`cli.conf`).
- **Reliability and Stability:** The CLI must handle network errors, invalid input, and unexpected server responses gracefully.

Technical choices To meet these requirements, the following technical decisions were made:

- **Programming language:** The CLI is implemented in **C3**, which allows low-level control while leveraging the nobuild build system.
- **Network library: libCurl** is used for HTTPS communication, providing robust and well-tested client-server interaction.
- **Terminal library: libTermbox** is utilized for rendering text-based graphics and handling keyboard input in a cross-platform terminal environment.
- **Build system:** The compilation is fully automated through the `build.c` script, which generates the executable (`transcli`) and ensures all dependencies are correctly linked.
- **Configuration handling:** A single configuration file (`cli.conf`) defines the server address. The build script reads this file and generates `gen.c3` to embed the server address at compile time, ensuring immutability at runtime.
- **Error handling:** The program includes validation of server responses and input data to maintain stability during execution.

3.2 API Communication

Requirements The CLI module must communicate reliably with the Transcendance server using secure protocols. The main requirements for API communication are:

- **Secure transmission:** All communication must be encrypted, using HTTPS for sensitive operations and WSS for real-time gameplay data.
- **Authentication:** The CLI must allow users to authenticate securely, using a token-based system.
- **Endpoint access:** The CLI should interact with specific server endpoints:
  - `/api/user` for user authentication, including 2FA validation.
  - `/PongSocket` to establish a WebSocket connection for the main game loop.
  - `/api/game` to register the user and their games in the server database.
- **Data format:** All requests and responses should use JSON, which must be correctly parsed and generated by the CLI.
- **Error handling:** Any communication errors, failed requests, or server responses must be logged for debugging purposes.

Technical choices To fulfill these requirements, the following technical decisions were made:

- **Protocol usage:**
  - **HTTPS POST** is used exclusively for user management operations such as registration, login, and 2FA validation. No GET requests are performed.
  - **WSS** is used for the real-time Pong game loop, ensuring low-latency, bidirectional communication.
- **Authentication:** A token-based system is implemented, where the token is sent in the HTTP header using `Authorization: Bearer <token>`.
- **JSON parsing and generation:** The CLI uses the built-in C3 library `std::encoding::json` for parsing JSON responses. A custom JSON generator was implemented with three methods:
  - `setValue` — sets a named value in a JSON object.
  - `set` — adds a value to a JSON array.
  - `next` — appends a comma to allow subsequent values in an array or object.

- **Error logging:** All errors, including failed requests or invalid server responses, are written to `windowLogs.txt` for review and debugging.
- **Library selection:** **libCurl** is used for HTTPS communication, while WSS is handled using a custom C3 implementation to integrate seamlessly with the main loop of the game.

3.3 Authentication

Requirements The CLI module must ensure secure and reliable user authentication to prevent unauthorized access. The main authentication requirements are:

- **Secure credentials:** User login information must be transmitted securely to the server.
- **Token-based authentication:** Once authenticated, a token must be provided and used for subsequent requests to authorize actions.
- **Two-factor authentication support:** The CLI must allow the server to enforce 2FA verification during login.
- **Integration with API endpoints:** Authentication must use the specific server endpoints for validation:
  - `/api/user/login` for username and password validation.
  - `/api/user/verify-email` for 2FA code verification.
- **Error handling:** Failed login attempts or invalid tokens must be logged and reported clearly to the user.

Technical choices To fulfill these requirements, the following technical decisions were made:

- **Login process:** The CLI sends a secure HTTPS POST request to `/api/user/login` with the user credentials.
- **Two-factor authentication:** If 2FA is required, the CLI sends the verification code to `/api/user/verify-email` for validation.
- **Token management:** Upon successful authentication, the server returns a token, which is stored in memory during the CLI session. This token is sent with every subsequent request in the HTTP header using `Authorization: Bearer <token>`.
- **Error reporting:** Any authentication failures, invalid tokens, or network errors are logged in `windowLogs.txt` and displayed in the CLI to inform the user.
- **Integration with main loop:** The token is automatically included in WSS connection requests for the Pong game, ensuring that all real-time actions are securely authorized.

3.4 Synchronization

Requirements The CLI module must ensure accurate and real-time synchronization with the server during Pong gameplay. The main requirements for synchronization are:

- **Real-time updates:** Game state must be updated continuously to reflect the positions of the ball, paddles, and scores.
- **Low latency:** Messages exchanged between client and server should be delivered with minimal delay to ensure a smooth gameplay experience.
- **Error handling and reconnection:** The client must be able to handle disconnections and resume communication with the server automatically.
- **Efficient data transfer:** Only necessary game data should be sent to reduce bandwidth usage and allow the CLI to parse JSON fast enough.
- **Client responsibilities:** The CLI is responsible solely for rendering the game in the terminal and capturing user input, while all game logic, collisions, and positions are handled by the server.

Technical choices To meet these requirements, the following technical decisions were made:

- **Main loop concurrency:** The CLI uses a concurrent main loop rather than traditional threads to maintain memory safety while handling multiple tasks simultaneously.
- **Message sending strategy:** - Messages are sent at each user action, except during game phases that require continuous updates, where messages are sent at every tick.
- **Data minimization:** Only essential game information is sent to the client to allow fast JSON parsing and maintain the server's update rate.
- **Automatic reconnection:** Reconnection is triggered by a specific message sent from the server, rather than by the underlying SocketIO implementation. This design simplifies concurrency in the main loop and provides a development workflow similar to that of Typescript's socketIO style.
- **Client-side responsibilities:** The CLI renders the game state in the terminal and captures key presses, while all collision detection, position updates, and game logic are performed server-side.

3.5 Cross Platform

Requirements The CLI module must be compatible with standard Linux terminals while allowing users to play against other players connected via web browsers. The main requirements for cross-platform support are:

- **Linux terminal compatibility:** The CLI should function correctly on all major Linux terminal emulators.
- **Server interoperability:** The CLI must be able to communicate with the server so that users can play against opponents using browsers (Firefox, Chrome, etc.).
- **Consistent input and rendering:** User input and game rendering must behave consistently across different terminal emulators.
- **Handling terminal limitations:** The CLI must provide a smooth gameplay experience despite the lack of some input events (e.g., KeyUp) in the terminal environment.

Technical choices To fulfill these requirements, the following technical decisions were made:

- **Terminal support:** The CLI runs exclusively under Linux but has been tested on multiple terminals including `gnome-terminal`, `blackbox`, and `alacritty`, ensuring consistent behavior.
- **Terminal library:** A modified version of **libTermbox**, a minimal ncurses-like library, is used to render the game and capture input. Modifications include RGB color support to enhance game graphics.
- **Input handling:** Since libTermbox does not provide KeyUp events, a system was implemented where:
  - The player moves continuously in one direction until the opposite arrow key is pressed, which changes direction.
  - Pressing the spacebar stops the player's movement.
- **Server interoperability:** The CLI communicates with the Transcendance server via WSS/HTTPS, allowing gameplay against users on browsers (Firefox, Chrome, etc.) without any major1 difference in behavior.
- **Consistency across terminals:** All tested Linux terminals behave identically thanks to libTermbox's abstraction, ensuring that rendering, input, and gameplay logic are uniform.

3.6 Documentation

Overview The documentation for the CLI module is entirely contained in this PDF. Its primary purpose is to provide the correctors with a comprehensive understanding of the module, its implementation, and its interaction with the server. It is not intended for end-users or external developers.

Structure The documentation is organized into clearly defined sections to facilitate navigation and comprehension:

- **Introduction:** Presents the objectives of the CLI module and its role within the Transcendance project.
- **User Manual:** Explains the build process and configuration required to compile and run the CLI.
- **Subject Requirements:** Details the requirements for the CLI application, API communication, authentication, synchronization, cross-platform support, and documentation itself.
- **Implementation:** Describes the code architecture, programming languages used, and external libraries integrated in the project.
- **Glossary:** Provides definitions for technical terms and acronyms used throughout the document.

Navigation and usability The document is designed to be self-contained and easy to navigate:

- Sections are organized logically, from general objectives to technical implementation details.
- Hyperlinks (via `hyperref`) allow quick access to glossary entries and references within the document.
- Code examples are included using verbatim blocks to illustrate important implementation details.

Purpose and audience This documentation is specifically tailored for the project correctors. It emphasizes the technical design decisions, implementation choices, and the workflow followed to develop the CLI module, ensuring that the correctors can fully understand the module's architecture and operation.

4 Implementation

4.1 Code Architecture

The CLI module is organized into multiple layers to separate concerns and ensure maintainability:

- **app/** contains the main application logic, user interface, and state management.
  - `api/` handles server communication via HTTPS and WSS.
  - `states/` contains individual pages (login, menu, game, 2FA, wait).
  - UI helpers such as `border.c3`, `box.c3`, `button.c3`, and `textbox.c3` manage display and input.
- **curl.c3i** wraps libCurl for simplified HTTP requests.
- **logger.c3** centralizes logging, including errors and events.
- **window.c3** wraps Termbox and implements the main loop for event handling and rendering.
- **termobx.c / termbox.c3 / utf8.c** provide cross-terminal rendering and input handling, including RGB colors and UTF-8 support.
- **main.c3** serves as the entry point, initializing the App structure and starting the main loop.

4.1.1 App structure

The `App` struct encapsulates all essential state and resources:

- `Curl* curlHandle` — handle for HTTP communication
- `States currentState` — the current active page
- `Window w` — window and mainloop handler
- `Socket s` — WebSocket connection
- `String secret` — user secret for authentication
- `String lobbyName` — current lobby
- `bool _needQuit` — exit flag
- `int userId` — user identifier
- `void*[] tmpMem, usz tmpMemP` — temporary memory pool

4.1.2 Main loop implementation

The main loop is implemented in `Window.mainloop()` to handle events concurrently, without threads, for memory safety:

- Processes input events (keyboard, mouse, resize) in a non-blocking loop
- Calls registered callbacks for `onKeyCbs`, `onMouseCbs`, `onResizeCbs`, and `onUpdateCbs`
- Updates the terminal display each tick via `tb::present()`
- Maintains a fixed loop rate with `thread::sleep_ms(10)`
- Executes `onQuitCbs` callbacks on exit

4.1.3 Concurrency model

Instead of using traditional threads, the CLI uses a **concurrent main loop**. This simplifies development and avoids memory safety issues while allowing simultaneous processing of events, rendering, and network communication.

4.2 Used Libraries

The CLI module relies on several external and internal libraries to provide networking, terminal rendering, and system-level functionalities.

- **nob.h** — Header provided by Tsoding to create build scripts.
  - Allows generating build scripts in C3 for the project compilation.
- **bob.h** — Internal helper library written specifically for this project.
  - Provides simplified helper functions such as `rm(char*)`, `clone(char*)`, and `make(char*)` to streamline file operations during compilation.
- **libCurl** — Used for all HTTPS communication with the server.
  - Handles POST requests for authentication, JSON payload transmission, and server queries.
  - Wrapped in `curl.c3i` to simplify integration with C3 and the CLI architecture.
- **Termbox** — Terminal library for rendering and input handling.
  - Provides cross-terminal support, RGB colors, and event handling (keyboard, mouse, resize).
  - Integrated via `window.c3`, `utf8.c`, and `termbox.c3` to implement the main loop and manage the user interface.

- **C3 Standard Library** — Utilized for core language features, JSON parsing, and basic system interactions.
  - Used for file operations, time management, sleep functions, and input/output handling.
  - Provides a JSON parser used in conjunction with `json.c3` to generate and parse JSON messages for API and WebSocket communication.

4.2.1 Integration notes

These libraries collectively enable the CLI to:

- Communicate securely and efficiently with the server.
- Render a responsive and visually consistent terminal interface.
- Maintain a structured build process that simplifies compilation and dependency management.
- Handle concurrent events safely without relying on traditional threading.

5 Glossary

CLI : Command-Line Interface, a text-based interface allowing users to interact with the program through the terminal.

HTTPS : HyperText Transfer Protocol Secure, used for secure communication between the CLI and the server via POST requests.

WSS : WebSocket Secure, used for real-time, bidirectional communication between the CLI and the server for the Pong game.

API : Application Programming Interface, a way for the programmer to communicate easily with the server.

Token : A string provided by the server after user authentication, used in subsequent requests via the HTTP header `Authorization: Bearer <token>`.

JSON : JavaScript Object Notation, a lightweight data-interchange format used for communication between the CLI and server.

EngineIO / SocketIO : Networking libraries and protocols used for real-time WebSocket communication between client and server.

libTermbox : A minimal ncurses-like library used for terminal rendering and input handling in the CLI, supporting RGB colors.

nob.h : Header provided by Tsoding to create build scripts in C3.

bob.h : Internal helper library written for this project, providing simplified file operations such as `rm`, `clone`, and `make`.

Build script : The script (`build.c`) that compiles the CLI, fetches dependencies, and generates `gen.c3` from `cli.conf`.

SERVER_ADDRESS : The server IP and port used by the CLI, set at compile time via `cli.conf`.

App : The main struct containing the state of the CLI, including the current page, window, socket, user info, and temporary memory.

Main loop : The loop implemented in `Window.mainloop()` that handles events, rendering, and updates in a concurrent manner without threads.

Curl : Library used to send HTTP requests from the CLI to the server, wrapped in `curl.c3i`.

Logger : Component responsible for logging errors and events, typically written in `logger.c3`.

Terminal emulator : A program that provides a text-based interface to interact with the CLI. Examples: `gnome-terminal`, `alacritty`, `blackbox`.

2FA : Two-Factor Authentication, a security mechanism requiring a code sent to the user to verify identity during login.

TUI : Text-based User Interface, used for displaying the game, menus, and interactive components in the terminal.

Conclusion

This document concludes the description and implementation details of the CLI module for the Transcendance project. All design choices, technical implementations, and requirements have been presented to ensure that the correctors fully understand the module's structure and functionality.

End of Document