Netsketch: A Collaborative Whiteboard

Assignment Report

Giorgio Grigolo

Contents

1 Introduction

This implementation of Netsketch, a collaborative whiteboard, is built entirely in Rust, a statically typed, memory-safe, idiomatic, systems programming language.

1.1 Project Structure

Netsketch was divided into three main components: the *server*, the *client*, and a *shared library*. The server is responsible for most of the business logic, such as managing the state of the whiteboard and broadcasting changes to all connected clients. The client is a graphical user interface that allows users to draw on the whiteboard and see the changes made by other users. The shared library contains the data structures and algorithms used by both the server and the client.

To conform with the Rust ecosystem's conventions, the above components are manifested as a Rust workspace, with the server and the client as separate crates, and the shared library as a library crate, initialized by the `cargo init --lib` command.

1.2 Libraries Used

A considerable effort has been made to use as few external libraries (or *crates*) as possible. The only crates used are

- `clap` for the command-line interfaces of the server and the client,
- thiserror for better error handling,
- `bincode` for encoding and decoding data structures into and from bytes,
- `macroquad` for the client's graphical user interface, and
- `tracing` for pretty server side logging.

All other functionality has been implemented from scratch, or by using the Rust standard library `std`.

2 System Design

It is clear, as specified, that this game is a client-server application. Upon reading the requirements of this project, it was immediately clear that most of the logic should take place on the server side, whereas the client, must be kept as simple as possible; it must send clear and concise instructions to the server. With this in mind, while planning the layout of this system, I concluded that I'd like to keep the client as light as possible, while letting the server doing most of the heavy lifting.

2.1 Shared Library: `ns-core`

In this section, I will describe the main features of the core of Netsketch, the library that is used by both the client and the server. I will also attempt to shed some light on the most important data structures in this library.

2.1.1 The Whiteboard

The central part of Netsketch is the whiteboard, or *canvas*, as I will refer to it from now on. It is defined as an array of *canvas entries*, and a monotonically increasing counter, to keep track of the order in which the actions were performed, as well as to serve as a current pointer to the last entry that was inserted into the canvas.

```
struct Canvas {
    entries: Vec<CanvasEntry>,
    counter: usize,
}
```

This brings us to introduce the `CanvasEntry`. The `CanvasEntry` stores the action's ID, the object drawn (the `CanvasElement`), the username of who drew it (the `author`), and a shown flag to be used by the client to determine whether the entry should be displayed or not.

```
struct CanvasEntry {
    id: usize,
    element: CanvasElement,
    shown: bool,
    author: String,
}
```

The `CanvasElement`, finally is just an enum that contains all the possible objects that can be drawn on the canvas, and the required information to draw them such as sets of coordinates, colour and so on. The objects supported are `Line`, `Rectangle`, `Circle`, and `Text`.

Note

The `Canvas` is further wrapped in another struct, which varies depending on the context in which it is used. In fact, the way it is mutated and accessed is different in the server and in the client. This will be discussed in the following sections.

]