clj-paint

Installation

Unzip the provided `clj-paint.zip` file to a location of your choice

Ensure you have Java available on your PATH to run the provided .jar file

User guide

Review the user guide stored at `doc/user-guide.pdf` to learn how to use clj-paint commands

Usage

Run from Jar

To run clj-paint execute the following in your terminal

```
java -jar clj-paint.jar
```

Run from source code

If you would like to run the code from source code you can use:

```
clojure -X:run-x
```

Compile a new jar

A new jar file can be compiled using:

```
clojure -X:uberjar
```

Run tests for the project

```
clojure -X:test:runner
```

Project structure

The project is comprised of four source files & supporting documents/tests

clj-paint.clj

This file contains the project entry point for generating jars via `:gen-class` and does the stateful job of placing the user into a command loop, this is separated out so that the underlying machinery can be tested

command.clj

This file contains the command routing and parsing required to turn string commands like `I 2 3` into something our program understands

We are using cli-matic to help validate and parse those commands along with functions that are organising our data before forwarding each command on to our main file image.clj

Within our clj-matic "router" we are also adding our new canvas states to a history atom, while this is not required to reach our immediate goals it maybe helpful in the future to implement undo

history.clj

This file is to support the history preservation operations to the canvas, at the moment the canvas states are just stored in memory but separating state operations out early makes it easier to keep the state at the edge of the system

image.clj

The main logic for the application is found here, for most functions we're making use of `clojure.core.matrix` to create new canvas states and we're using `the-flood` version 1.1 to handle the flood filling behaviour

Test structure

Example based tests are specified in `resources/examples.txt` tests are seperated by `- - - -`

lines that start with `>` are considered input commands, many can be executed in order as they would be written by the user inside the application

The lines below `>` are considered the expected outputs for printing the canvases

When tests are run the examples file is read and we dynamically run the user commands from the file and then compare our program's result with our expected result in the .txt file

Lots of little bugs were found as part of this process so confidence is quite high for time invested but we could always implement other kinds of testing

Project improvements

Immutability concerns

At the moment the canvas state is fully immutable which means after extended use of the application would theoretically exceed memory constraints, we could start removing history from the beginning of the atom after a certain amount of changes have been persisted to help prevent this

Slow flood fill

Flood fill is implemented from a library to help speed up development but on larger canvas sizes this can freeze the application, to make this apparent clj-paint now prints `:OK` when work has finished so the user can tell if a command is taking longer than expected

If slowness becomes a problem we could look at implementing a more efficient flood fill function something similar to https://silven.nu/posts/4-parallel-flood-fil.html to help improve performance

If we can improve this upstream we can potentially get away with making no change to our application beyond bumping the flood fill library version

Writing to disk

Writing state out to disk would allow users to save and load previously made images from clj-paint

License