An algorithmics and data structures project:

Vector Text-based editor

2022-2023

Matthéo PHO
Julien BOUDJEDID

PARIS PANTHÉON-ASSAS UNIVERSITÉ

Table des matières

1. Introduction

Two months ago, we were tasked a project that combines the power of the C programming language with the creativity of vector and shapes drawing. In this text-based project, we explore the realm of computer graphics, allowing users to create and manipulate geometric shapes using simple text commands.

Using this newly learnt programming language, we were asked to build an intuitive and interactive drawing tool. Users can create lines, rectangles or polygons and display them on a plot using simple text commands.

This project is an invaluable learning experience for us as students, as it allows us to get familiar with the fundamental concept of structures and arrays. These data structures play a vital role in storing and displaying information, such as shapes or coordinates. Furthermore, we need to learn about different shapes algorithms to display them on the plot.

2. Functionalities

Starting the project, we were given several specifications as to what was to be achieved. Our final product answer pretty much all those specifications and embark a few additional functionalities.

- Specified functionalities.

a. Creation of different shapes.
b. Plot shapes on a fixed-size board.
c. List all shapes.
d. EDIT shape values.
e. Delete shapes by ID.
f. Erase all shapes.

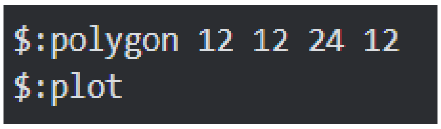
- Additional functionalities

g. Help command for available commands.
The help command gives access to information about any command such as its semantic and the variables to determine.

h. Read files read and execute files of commands.
User is able to read files and write them manually.

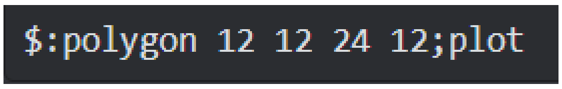
i. multiple instruction and includes a verbose mode.
Allow the user to execute multiple instructions in one line of command.

Instead of:



```
$:polygon 12 12 24 12
$:plot
```

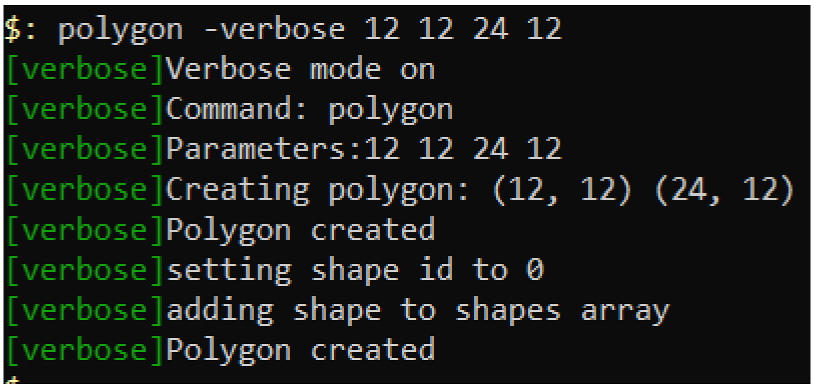
We have:



```
$:polygon 12 12 24 12;plot
```

Verbose mode gives us information about the ongoing process.

Example:



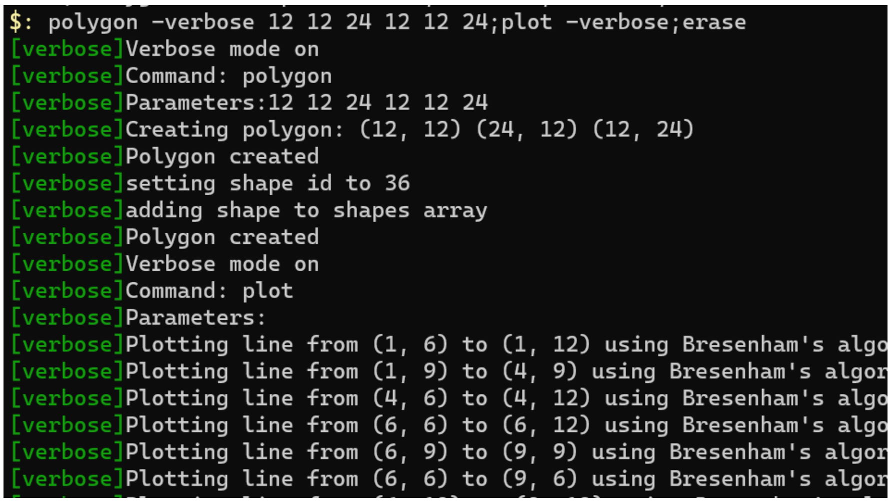
```
$: polygon -verbose 12 12 24 12
[verbose]Verbose mode on
[verbose]Command: polygon
[verbose]Parameters:12 12 24 12
[verbose]Creating polygon: (12, 12) (24, 12)
[verbose]Polygon created
[verbose]setting shape id to 0
[verbose]adding shape to shapes array
[verbose]Polygon created
```

j. Add colors to the shapes.

Each color corresponds to a shape type, making the plot more readable and more alive.

3. Technical presentation

A. Command Parsing

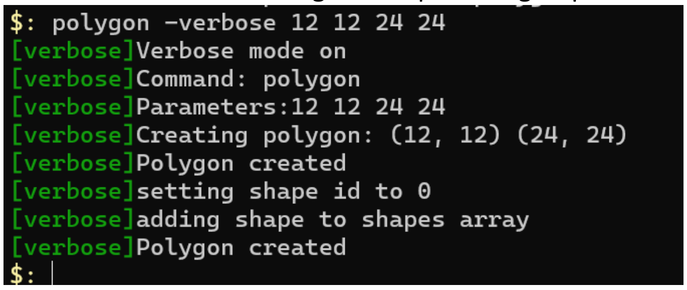


```
$: polygon -verbose 12 12 24 12 12 24;plot -verbose;erase
[verbose]Verbose mode on
[verbose]Command: polygon
[verbose]Parameters:12 12 24 12 12 24
[verbose]Creating polygon: (12, 12) (24, 12) (12, 24)
[verbose]Polygon created
[verbose]setting shape id to 36
[verbose]adding shape to shapes array
[verbose]Polygon created
[verbose]Verbose mode on
[verbose]Command: plot
[verbose]Parameters:
[verbose]Plotting line from (1, 6) to (1, 12) using Bresenham's algo
[verbose]Plotting line from (1, 9) to (4, 9) using Bresenham's algor
[verbose]Plotting line from (4, 6) to (4, 12) using Bresenham's algo
[verbose]Plotting line from (6, 6) to (6, 12) using Bresenham's algo
[verbose]Plotting line from (6, 9) to (9, 9) using Bresenham's algor
[verbose]Plotting line from (6, 6) to (9, 6) using Bresenham's algor
```

The parseCommand() function in commands.c is responsible for parsing user-entered commands. It takes a string as input and returns a pointer to a Command structure containing the command name, parameters, and the number of parameters.

The function first extracts the command name and then iterates through the input string to parse the parameters as integers. To handle multiple commands separated by semicolons, the main.c file uses the strtok() function to split the input string into tokens.

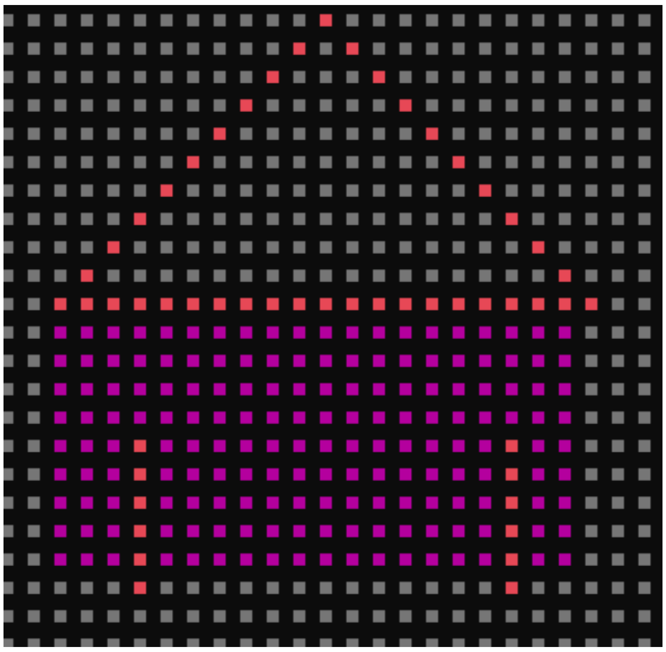
This makes it possible to stack multiple commands on one line, allowing for more



```
$: polygon -verbose 12 12 24 24
[verbose]Verbose mode on
[verbose]Command: polygon
[verbose]Parameters:12 12 24 24
[verbose]Creating polygon: (12, 12) (24, 24)
[verbose]Polygon created
[verbose]setting shape id to 0
[verbose]adding shape to shapes array
[verbose]Polygon created
$:
```

complex drawings with a single input. However, we also have some special parameters, such as '-verbose', that can be used on any command to show the inner workings of the program.

B. Shape Drawing



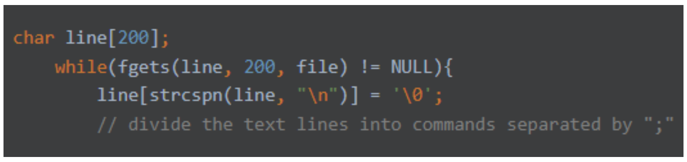
The plot() function in shapes.c is responsible for drawing the shapes on the board. It first initializes an empty board and then iterates through the list of shapes to draw each one depending on its type.

For lines, the function uses Bresenham's line algorithm to plot the line points on the board.

C. reading files

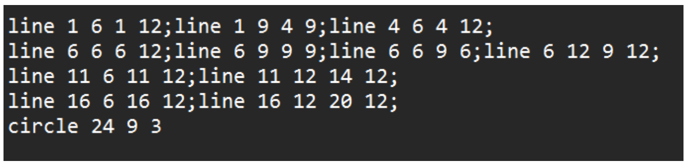
using our built-in function fileCommand we read instructions from .ccp files using fopens

we then divide text into commands like in the terminal:

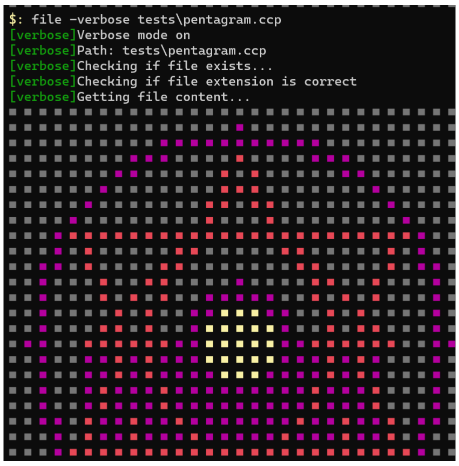


```c
char line[200];
    while(fgets(line, 200, file) != NULL){
        line[strcspn(line, "\n")] = '\0';
        // divide the text lines into commands separated by ";"
```

example ccp file content:



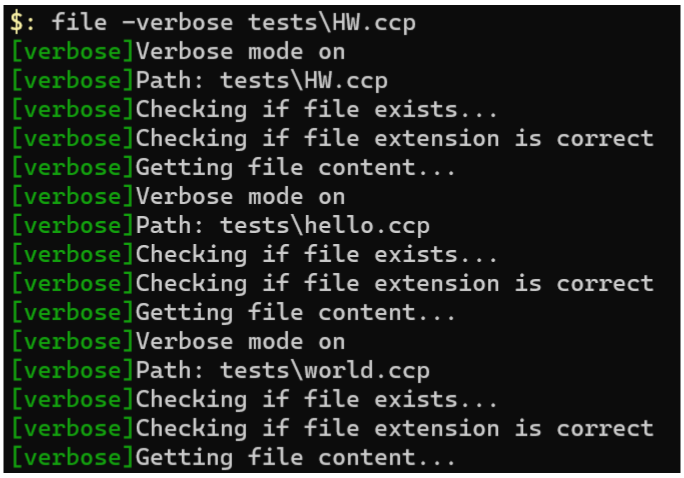
```
line 1 6 1 12;line 1 9 4 9;line 4 6 4 12;
line 6 6 6 12;line 6 9 9 9;line 6 6 9 6;line 6 12 9 12;
line 11 6 11 12;line 11 12 14 12;
line 16 6 16 12;line 16 12 20 12;
circle 24 9 3
```



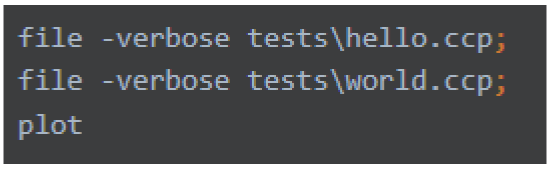
```
$: file -verbose tests\pentagram.ccp
[verbose]Verbose mode on
[verbose]Path: tests\pentagram.ccp
[verbose]Checking if file exists...
[verbose]Checking if file extension is correct
[verbose]Getting file content...
```

D. recursive file calling

Here is an example of what happens when you call a file that calls other files:



```
$: file -verbose tests\HW.ccp
[verbose]Verbose mode on
[verbose]Path: tests\HW.ccp
[verbose]Checking if file exists...
[verbose]Checking if file extension is correct
[verbose]Getting file content...
[verbose]Verbose mode on
[verbose]Path: tests\hello.ccp
[verbose]Checking if file exists...
[verbose]Checking if file extension is correct
[verbose]Getting file content...
[verbose]Verbose mode on
[verbose]Path: tests\world.ccp
[verbose]Checking if file exists...
[verbose]Checking if file extension is correct
[verbose]Getting file content...
```

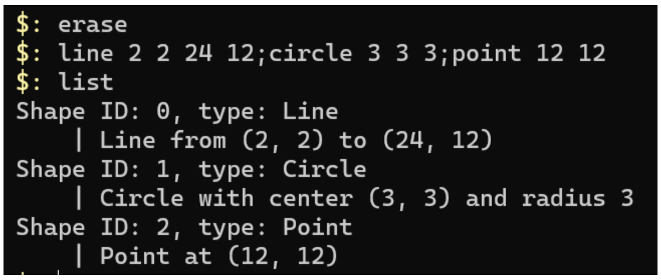


```
file -verbose tests\hello.ccp;
file -verbose tests\world.ccp;
plot
```

E. Modifying values of already placed objects.

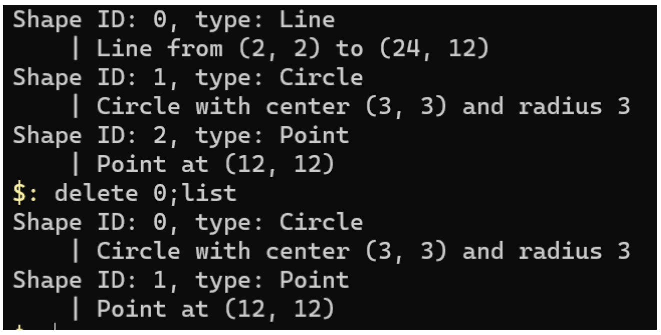
a. Accessing objects

Using our list function, we can easily access all objects on the board:



```
$: erase
$: line 2 2 24 12;circle 3 3 3;point 12 12
$: list
Shape ID: 0, type: Line
    | Line from (2, 2) to (24, 12)
Shape ID: 1, type: Circle
    | Circle with center (3, 3) and radius 3
Shape ID: 2, type: Point
    | Point at (12, 12)
```

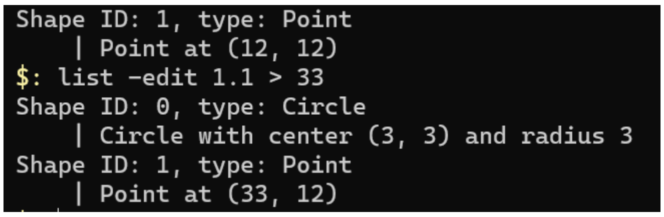
b. the delete function.



```
Shape ID: 0, type: Line
    | Line from (2, 2) to (24, 12)
Shape ID: 1, type: Circle
    | Circle with center (3, 3) and radius 3
Shape ID: 2, type: Point
    | Point at (12, 12)
$: delete 0;list
Shape ID: 0, type: Circle
    | Circle with center (3, 3) and radius 3
Shape ID: 1, type: Point
    | Point at (12, 12)
```

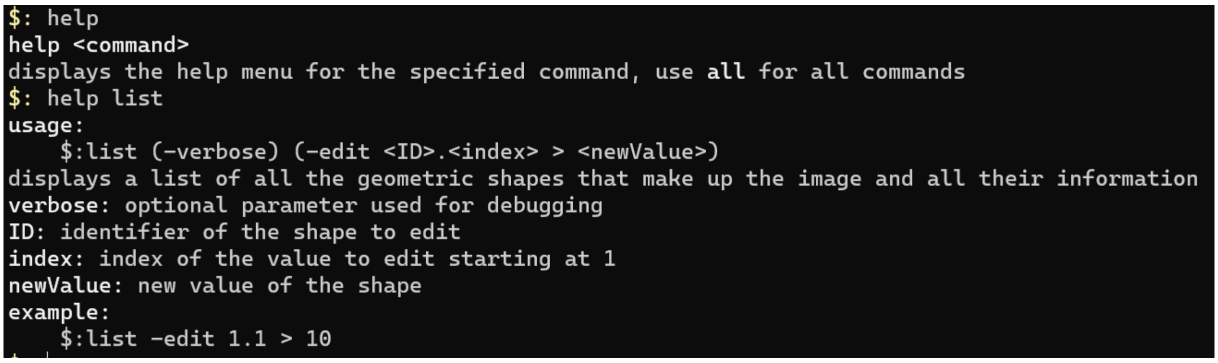
c. Modification of values

Modify values using the special -edit parameter for the list command.



```
Shape ID: 1, type: Point
    | Point at (12, 12)
$: list -edit 1.1 > 33
Shape ID: 0, type: Circle
    | Circle with center (3, 3) and radius 3
Shape ID: 1, type: Point
    | Point at (33, 12)
```

F. A useful help menu for each command



```
$: help
help <command>
displays the help menu for the specified command, use all for all commands
$: help list
usage:
    $:list (-verbose) (-edit <ID>.<index> > <newValue>)
displays a list of all the geometric shapes that make up the image and all their information
verbose: optional parameter used for debugging
ID: identifier of the shape to edit
index: index of the value to edit starting at 1
newValue: new value of the shape
example:
    $:list -edit 1.1 > 10
```

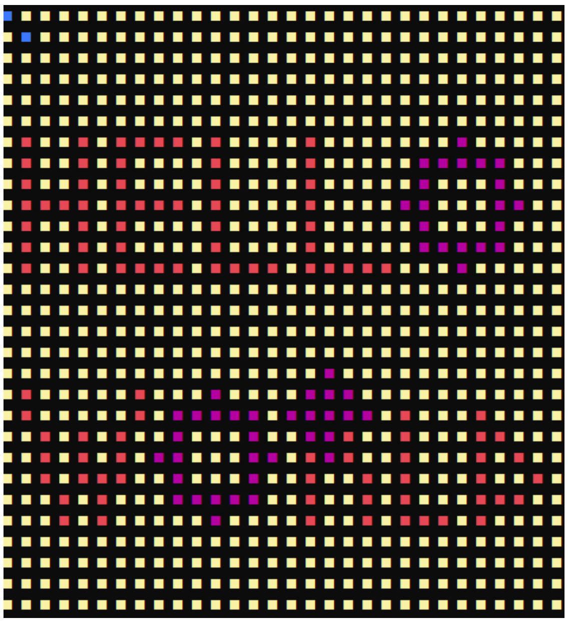
hint: typing 'help all' returns a list of all the usable commands.

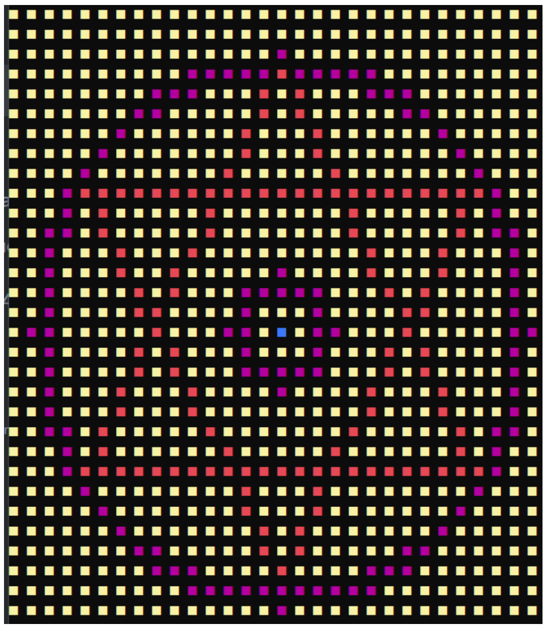
4. Challenges Faced

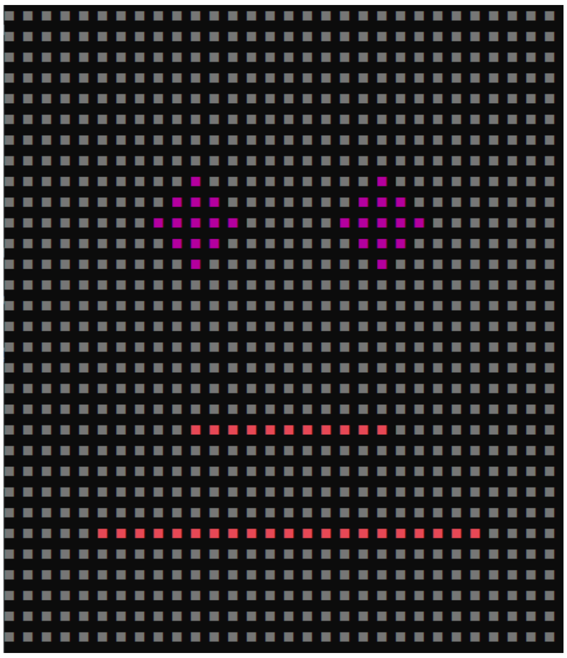
- Handling command input and parsing: Implementing a command parser that can handle variable-length commands and parameters was challenging.
- Drawing shapes on the board: Implementing the logic for drawing different shapes on the board, especially lines and polygons, required a good understanding of geometry and algorithms.
- Memory management: Ensuring that memory is allocated and freed correctly for commands and shapes to avoid memory leaks.
- Dealing with alternative user inputs: sometimes the user will enter something unpredictable, or something incorrect, such as a polygon with only 1 coordinate, or a dot with characters instead of coordinates, we spent some time trying to understand how to fix this.
- Loss of PC: You may have noticed that this is completely different from our first filing, this is due to the fact that Julien had advanced a lot on the project on his local machine, and hadn't pushed his branch to the github due to lack of internet. However, before he was able to do so, his computer was stolen (email was sent to teacher to provide the procès verbal). As thus, during the time before he was able to get his PC back, he, and Mattheo, decided to restart the project from scratch, with all the new knowledge they had gained. Overall, it turned out file.

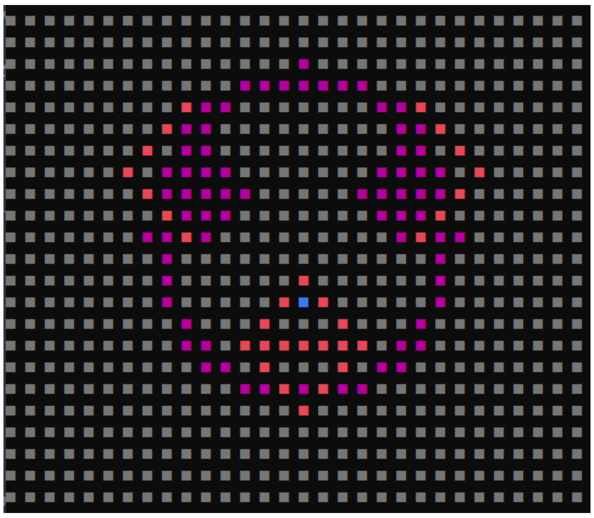
5. Results

You can access the demo file by using the command line: file tests/demo.ccp









6. Conclusion

In conclusion, our project has provided us with valuable lessons on multiple fronts. First and foremost, we have gained significant technical knowledge and expertise. Through the development of this project, we deepened our understanding of the C programming language, developed our skills in working with structures and arrays, and learned about shape algorithms.

Additionally, this project taught us about work organization and communication within a team. Collaborating effectively was crucial to the success of our project. We learned the importance of clear and concise communication, sharing ideas, and dividing tasks efficiently. Taking each other’s strength and weaknesses into account was an important point in creating a cohesive final product.

Furthermore, time management played a pivotal role throughout the project. Our academic commitments with the project's demands required careful planning and disciplined time management skills, especially with the DEs coming up in May.

Julien: Parsing inputs and dealing with parameters has been especially challenging and interesting to do, I’ve gained a lot more knowledge on how CLI’s work, and this has inspired me to start building my own, for a project I’ve been working on part time. I will definitely be using the knowledge I gained from this project to implement a similar CLI instruction parsing method. I really wanted to implement a file printing function, in order to create .png pixel art files from the board, and being able to edit the board size from the CLI, however, we had to prioritize the rest of our academics, and it might be something I implement in my free time later. Losing my PC has also been eye opening in regards to projects, as I now know to always push/commit before ending my session.

You can find the entire code and readme on Julien’s github: https://github.com/makethpanda/cleanCpainter