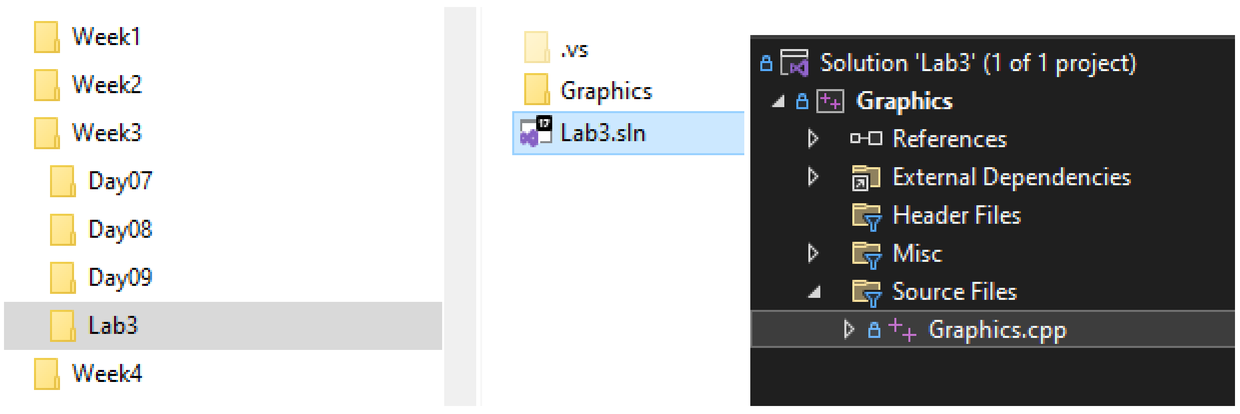
COP2334

PG2 – LAB: GRAPHICS LIBRARY

CONTENTS

SETUP

A **C++ console application** has been provided for you in your **GitHub repo**. **Use the provided solution.**



Lab Video

Here's a video showing what the lab could look like when completed:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EVDHtkw5vQJOjtxJYsiNIuUBSQSMkVtJZLwTLRpMf0t5nQ?e=Wtn7Id

How should you proceed?

The typical process for approaching the labs is…

Work on each part in order.

1. Watch the lecture video for Part A.
    a. The lectures have ***optional*** coding challenges. If you have time, then attempt them because they will help explain what you need to do for the lab.
2. Start working on Part A for the lab (Parts A-1 through A-3).
3. Once done with Part A, move on to Part B and repeat the steps above.

What to do if you need help?

Don't struggle for too long (more than 1 hour). Reach out for help to get your questions answered.

- Take advantage of the **open lab sessions** (they are in the evening so check the announcement channel in Discord for the zoom link). Here is the schedule: https://fullsailedu.sharepoint.com/:x:/s/emergingtechstu/EbBgblouNORLsms4V4hpAYgB0Mejv59w8IVGADkVgfJW8Q?e=0l2svR
- Ask your questions in the **PG2 Discord channels**. There will be instructors and students online who can help.
- Check the availability of **Full Sail tutors**. (https://discord.com/invite/8nV8PBqq7z)

Committing your code

You MUST commit and push your code to GitHub after completing each section. For instance, part A of this lab has 3 sections. You must commit and push after completing each section which would mean you end up with 3 separate commits for part A.

If you do not commit and push for every section, you could be deducted up to 15 points for the lab.

PART A

Lecture Videos for Part A

Day 07: OOP – Classes Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EdFrtn5AIYRIibjiR9fEto8Bm4ksCe0E48BdsqnoVE-21Q?e=df02RG

Part A-1: Point2D struct

Structures in C++

Add a **Point2D** struct to the **Graphics** project. Right-click the project, select Add->New Item. Name the file Point2D.h.

Add 2 **fields**: x and y. The type of these fields is int. (They need to be visible outside of the struct)

Add a constructor that takes 2 int parameters. Use these to initialize x and y.

Make sure to commit + push after completing the section.

Part A-2: Shape class

Add a **Shape** class to the **Graphics** project.

Add **2 fields with getters/setters**: a **Point2D** called **startPt** and a **ConsoleColor** called **color**.

Add **2 constructors**:

- First constructor takes 2 parameters: a Point2D and a ConsoleColor. Set the data with these parameters.
- Second constructor takes 3 parameters: 2 ints for the x and y and a ConsoleColor. Use the x and y parameters to initialize the startPt. Use the ConsoleColor parameter to set the color.

Add a **draw** method.

- Set the background color.
- Move the cursor to the Point2D position.
- Print a space.
- Reset the color in the console. (call the reset method on the Console class).

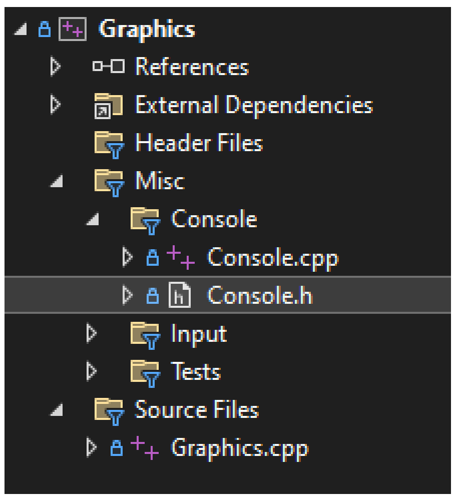
Make sure to commit + push after completing the section.

Part A-3: draw Shape menu

In **main** (which is in **Graphics.cpp**), add code to case 1 of the menu switch. Generate a random Point2D with an x,y anywhere in the console. Use `getWindowWidth()` and `getWindowHeight()` which are part of the Console class that is provided in the project (see screenshot below). Read the **Console.h** file to discover how to call it. Use that point to create a Shape instance with any color you want. Call draw on the Shape instance.

Example Shape:

Location of the Console.h file.



Make sure to commit + push after completing the section.

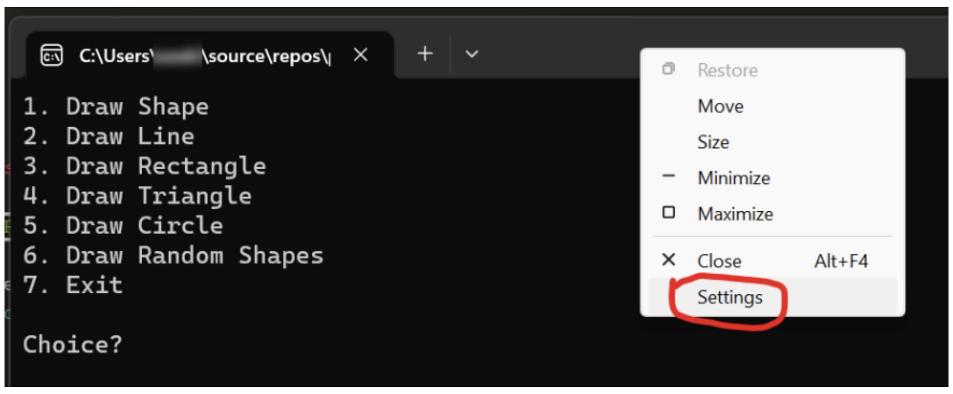
HELP! My Shapes are not drawing correctly!

Your system might be using the Windows Terminal which prevents some backend resizing of the console window.

If your lab looks like the first screenshot when running, then try changing the settings as shown in the screenshots:

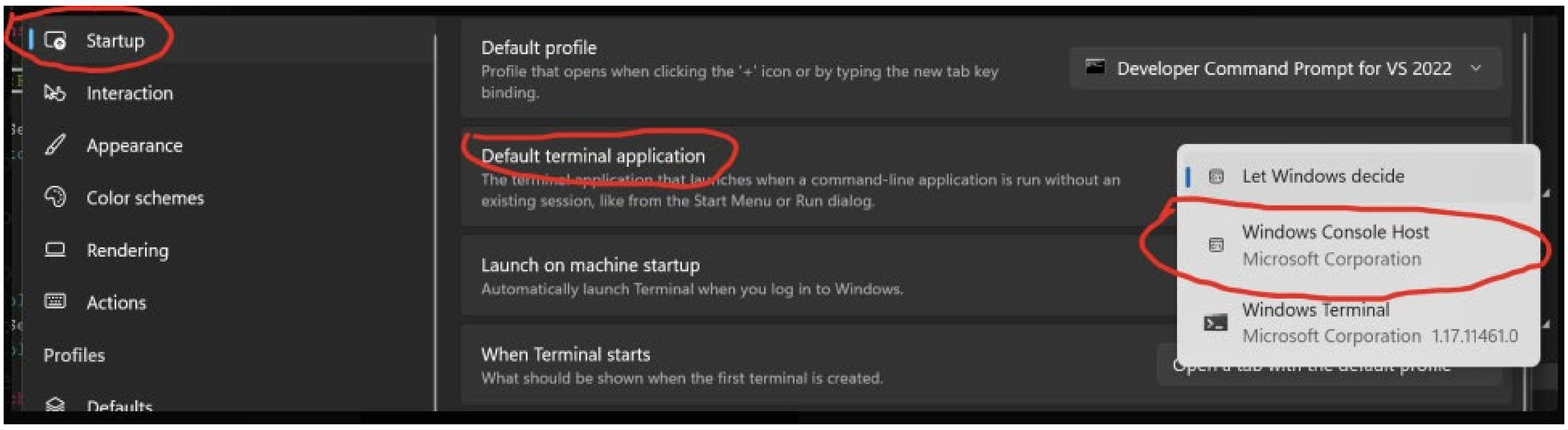
Here's also a page that describes the different ways to change the windows setting. NOTE: change the setting to use "Windows Console Host".

https://winaero.com/windows-11-change-console-to-windows-terminal-or-command-prompt/



```
1. Draw Shape
2. Draw Line
3. Draw Rectangle
4. Draw Triangle
5. Draw Circle
6. Draw Random Shapes
7. Exit

Choice?
```



Default profile
Profile that opens when clicking the '+' icon or by typing the new tab key binding.
Developer Command Prompt for VS 2022

Default terminal application
The terminal application that launches when a command-line application is run without an existing session, like from the Start Menu or Run dialog.

Launch on machine startup
Automatically launch Terminal when you log in to Windows.

When Terminal starts
What should be shown when the first terminal is created.

PART B

Lecture Videos for Part B

Day 08: OOP – Inheritance and Polymorphism Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/Ef9tdBd7hpRCpGzth-0WtXcB9yIBttB04SbyP4vgqkhuWw?e=lbO6Zq

Part B-1: Line class

Add a **Line** class to the **Graphics** project. The Line class should **derive from Shape**.

Add 1 Point2D **field** called endPt. Add getter/setter methods for the field.

Add a **constructor** that takes a start Point2D, end Point2D, and console color. Make sure to call the base constructor. Don't duplicate what the base constructors do.

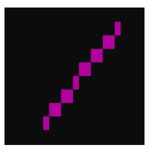
Implement the pseudocode for the **PlotLine** (see the pseudo-code for PlotLine below). NOTE: you'll need to implement the Plot method – it should move the cursor to the x,y position and prints a space.

Override the draw method of the Shape class (that means you need to mark the base as virtual). Do not call the base. Instead, set the background color, call PlotLine, then reset the color.

```
PlotLine(x0, y0, x1, y1)
    dx = abs(x1 - x0)
    sx = x0 < x1 ? 1 : -1
    dy = -abs(y1 - y0)
    sy = y0 < y1 ? 1 : -1
    error = dx + dy

    while true
        Plot(x0, y0)
        if x0 == x1 && y0 == y1 break
        e2 = 2 * error
        if e2 >= dy
            if x0 == x1 break
            error = error + dy
            x0 = x0 + sx
        end if
        if e2 <= dx
            if y0 == y1 break
            error = error + dx
            y0 = y0 + sy
        end if
    end while
```

In **main** (which is in **Graphics.cpp**), add code to case 2 of the menu switch. Generate 2 random Point2D points with an x,y anywhere in the console. Use those points to create a Line instance with any color you want. Call draw on the Line instance.
Example Line:



Make sure to commit + push after completing the section.

Part B-2: Rectangle class

Add a **Rectangle** class to the **Graphics** project. The Rectangle class should **derive from Shape**.

Add 2 int **fields** with getters/setters. Name the fields width and height.

Add 1 vector<Line> field called lines.

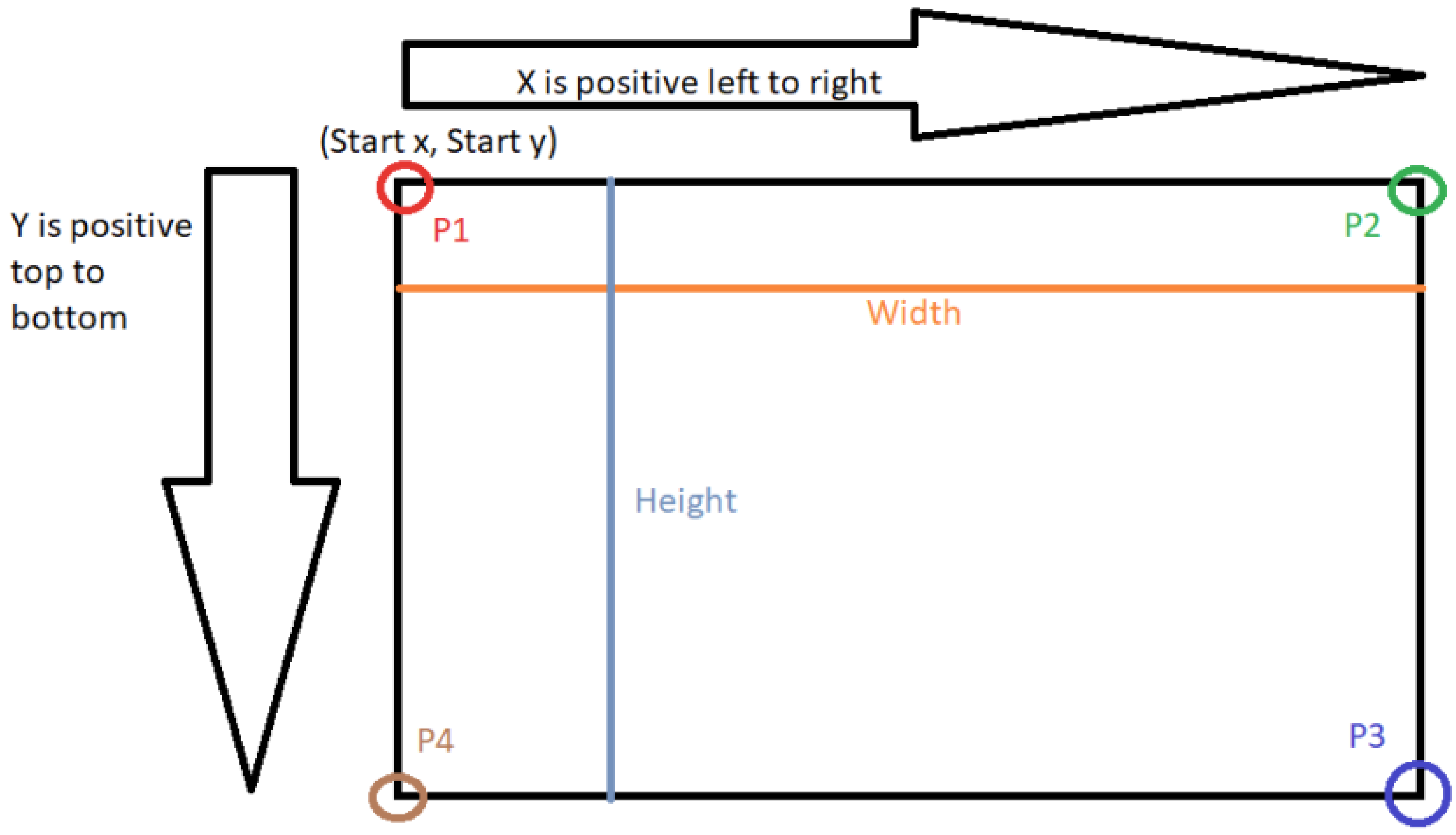
Add 1 constructor with the following parameters: width, height, startPt, color. Pass the startPt and color to the base constructor. Use width and height to set the fields. The constructor should create 4 Lines and add them to the lines field.

The 4 Lines:

- top left to top right
- top right to bottom right
- bottom left to bottom right
- top left to bottom left

Override the draw method of the Shape class (that means you need to mark the base as virtual). Do not call the base. Instead, call the draw method of each Line in the lines vector.

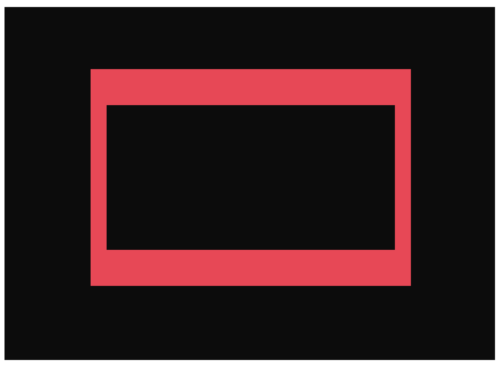
The rectangle constructor should be given a start point, a width, and a height. Use that information to create the 4 points of the rectangle which you'll use to create the 4 lines of the rectangle.



In **main** (which is in **Graphics.cpp**), add code to case 3 of the menu switch.

- Generate a random Point2D point with an x,y anywhere in the console. This point will be the top-left position of the Rectangle.
- Calculate a random width and height – ensure that it will NOT extend the Rectangle beyond the bounds of the console.
- Use the point, width, and height to create a Rectangle instance with any color you want.
- Call draw on the Rectangle instance.

Example output:



Make sure to commit + push after completing the section.

Part B-3: Triangle class

Add a **Triangle** class to the **Graphics** project. The **Triangle** class should **derive from Shape**.

Add 2 Point2D **fields** with getters/setters. Name the fields p2 and p3.

Add 1 vector<Line> **field** called lines.

Add 1 **constructor** with the following parameters: p1, p2, p3, color. Pass the p1 and color to the base constructor. Use p2 and p3 to set the data. The constructor should create 3 Lines connecting the points and add them to the lines field.

The 3 Lines:

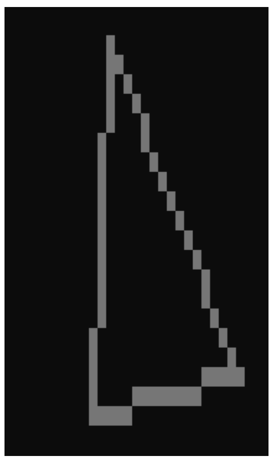
- P1 to p2
- p2 to p3
- p3 to p1

Override the draw method of the Shape class (that means you need to mark the base as virtual). Do not call the base. Instead, call the draw method of each Line in the lines vector.

In **main** (which is in **Graphics.cpp**), add code to case 4 of the menu switch.

- Generate 3 random Point2D points with an x,y anywhere in the console.
- Use the points to create a Triangle instance with any color you want.
- Call draw on the Triangle instance.

Example Triangle:



Make sure to commit + push after completing the section.

Part B-4: Circle class

Add a **Circle** class to the **Graphics** project. The Circle class should **derive from Shape**.

Add 1 int **field** with a getter/setter. Name the field radius.

Add 1 **constructor** with the following parameters: radius, startPt, color. Pass the startPt and color to the base constructor. Use radius parameter to set the radius data.

Implement the pseudocode for the **DrawCircle, DrawCirclePoints,** and **Plot** methods. (see the pseudo-code below). NOTE: Plot is a method that moves the cursor to the x,y position and prints a space. **Override the draw method** of the Shape class (that means you need to mark the base as virtual). Do not call the base. Instead, set the background color, call DrawCircle, then reset the color.

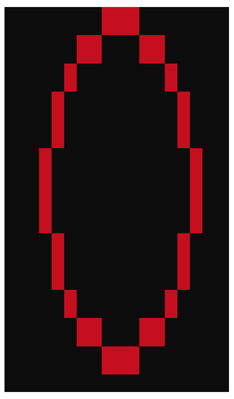
```
DrawCirclePoints(xc, yc, x, y)
    Plot(xc+x, yc+y)
    Plot(xc-x, yc+y)
    Plot(xc+x, yc-y)
    Plot(xc-x, yc-y)
    Plot(xc+y, yc+x)
    Plot(xc-y, yc+x)
    Plot(xc+y, yc-x)
    Plot(xc-y, yc-x)
end DrawCirclePoints

DrawCircle(xc, yc, r)
    x = 0, y = r
    d = 3 - 2 * r
    DrawCirclePoints(xc, yc, x, y)
    while y >= x
        x = x + 1
        if d > 0
            y = y - 1
            d = d + 4 * (x - y) + 10
        else
            d = d + 4 * x + 6
        DrawCirclePoints(xc, yc, x, y)
    end while
end DrawCircle
```

In **main** (which is in **Graphics.cpp**), add code to case 5 of the menu switch.

- Generate a random Point2D point with an x,y anywhere in the console. This point will be the center position of the Circle.
- Calculate a random radius – ensure that it will NOT extend the Circle beyond the bounds of the console.
- Use the point and radius to create a Circle instance with any color you want.
- Call draw on the Circle instance.

Example Circle:



Make sure to commit + push after completing the section.

PART C

Lecture Videos for Part C

Day 09: OOP – Miscellaneous Concepts Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EWWOJxQclapDj5sSBKq9jGUBIGSNlYUszhmtJiKwPsAdgw?e=JwEFTt

Part C-1: ShapeFactory

Create a class called ShapeFactory. Add 7 **static** methods: a method for creating a random point, a method for creating a random color, and methods for creating each Shape type. The methods that create the random Shapes should use `make_unique` and return a `unique_ptr`.

RandomPoint should return a random Point2D struct.

RandomColor should return a random ConsoleColor.

RandomShape should return a random Shape.

RandomLine should return a random Line.

RandomRectangle should return a random Rectangle.

RandomTriangle should return a random triangle.

RandomCircle should return a random Circle.

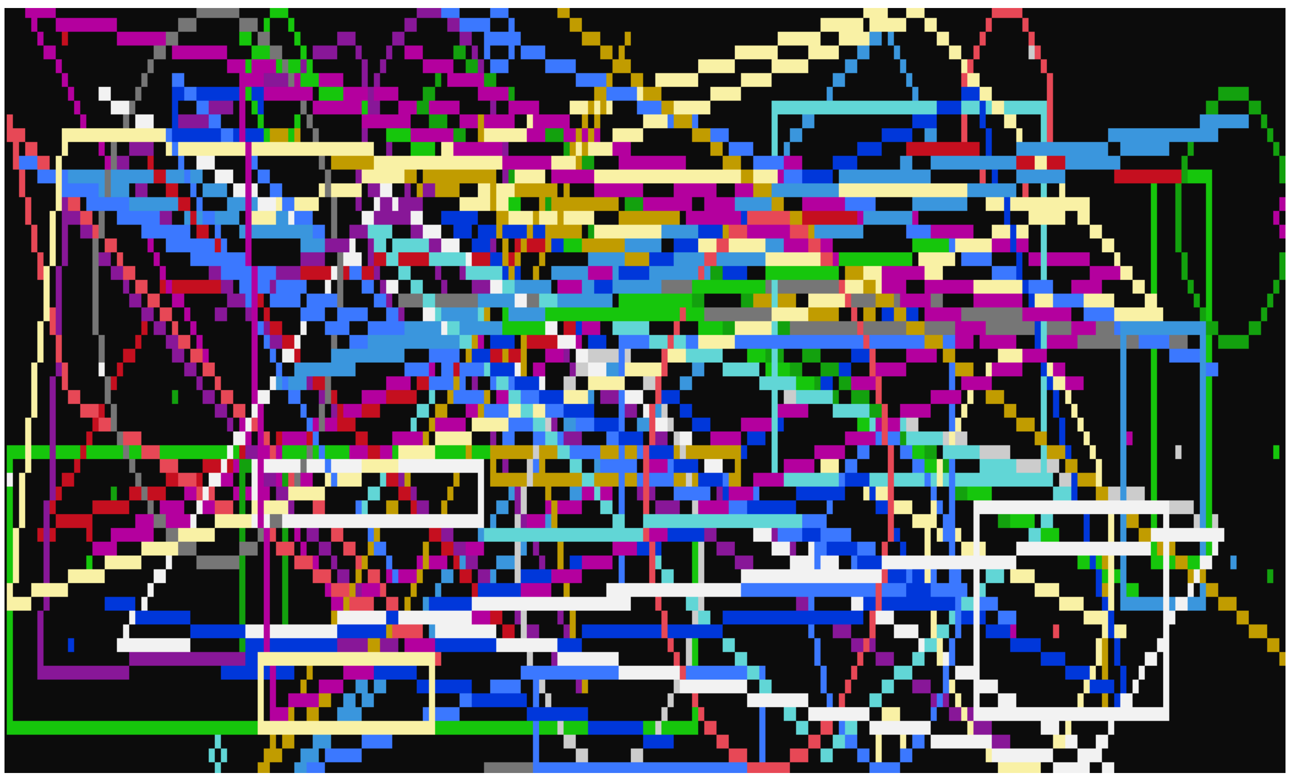
Make sure to commit + push after completing the section.

Part C-2: Random Shapes

In **main** (which is in **Graphics.cpp**), add code to case 6 of the menu switch.

- Create a vector variable that holds unique_ptr of Shapes.
- Create 100 random Shapes and add them to the vector.
    - Randomly pick which type of Shape to create (Shape, Line, Rectangle, Triangle, Circle).
    - Create the instance according to its Shape. (call the appropriate method of the ShapeFactory)
- After this loop, loop over the Shapes vector and call draw on each Shape.

Example output:



Make sure to commit + push after completing the section.