CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

Overview

The goal of this project is to practice working with interfaces and other concepts in Go by means of drawing shapes to the screen.

The shapes that could be drawn are a rectangle, triangle, or a circle. Each of those are structs in this Go project, and they must implement the **`geometry`** interface. The shapes will be drawn on a display, which is another struct that contains the pixel information along with the dimensions of the image. It must implement the **`screen`** interface, which consists of methods that interact with a display.

The program starts by asking the user for the dimensions of the display. It then runs a loop that allows the user to draw shapes, until they enter "X" to stop drawing. Finally, it asks the user to enter the name of the file in which the resulting figure should be saved, and writes the pixel information and the header to a PPM file, before exiting the program.

PPM Image Format

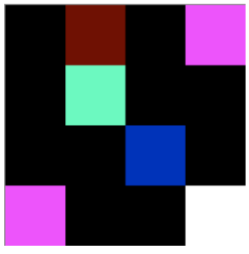
There are many image formats you are no doubt familiar with: JPG, PNG, etc. The advantage of PPM is that the file format is human-readable, so you can open PPM files in a text editor. This makes it easier to write programs that manipulate the images, and it also makes it easier to debug your output because you can simply open the image file in a text editor and "see" what's wrong.

The **PPM (or Portable Pix Map) image format** is encoded in human-readable ASCII text. The formal image specification can be found at https://netpbm.sourceforge.net/doc/ppm.html.

Here is a sample PPM file, representing a very small 4x4 image:

```
P3
4 4
255
0  0  0   100 0  0       0  0  0    255   0 255
0  0  0    0 255 175     0  0  0     0    0  0
0  0  0    0  0  0       0 15 175    0    0  0
255 0 255  0  0  0       0  0  0    255  255 255
```

Here is what this image looks like (magnified). Notice it consists of 16 pixels, laid out in 4 rows with 4 pixels per row:



CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

You can think of an image as having two parts, a *header* and a *body*. The **header** consists of information about the image such as width and height, GPS location, time, date, etc..

For PPM images, the header is very simple, and has only 4 entries:

```
P3
4 4
255
```

P3 is a "magic number" that indicates what type of PPM image this is (full color, ASCII encoding). For this assignment it will always be the string “P3”. The next two values, 4 4, represent the width and height of the image. More accurately from a programming perspective, the number of pixels in one row is the width and the number of rows in the image is the height. The final value, 255, is the maximum color depth for the image. For images in “P3” format, the depth can be any value in the range 0 – 255, inclusive.

The **image body** contains the pixel data, i.e. the color of each pixel in the image. For the image shown above, which is a 4x4 image, we have 4 rows of pixel data:

```
0  0  0   100 0  0       0  0  0    255   0 255
0  0  0    0 255 175     0  0  0     0    0  0
0  0  0    0  0  0      0 15 175    0    0  0
255 0 255  0  0  0       0  0  0    255  255 255
```

First, notice the values range from 0 to the maximum color depth (in this case 255). Second, notice that each row contains exactly 12 values, with at least one space between each value. Why 12? This is because each row contains 4 pixels, but each pixel in PPM format consists of 3 values: the amount of RED, the amount of GREEN, and the amount of BLUE. This is more commonly known as the pixel’s RGB value. Black, the absence of color, has an RGB value of 0 0 0 — the minimum amount of each color. White, the presence of all colors, has an RGB value of depth depth depth — the maximum amount of each color. As shown above, notice the first pixel in the 4x4 image is black, and the last pixel is white.

In general a pixel’s RGB value is the mix of red, green, and blue needed to make that color. For example, here are some common RGB values, assuming a maximum color depth of 255:

Yellow: 255 255 0

Maroon: 128 0 0

Navy Blue: 0 0 128

Purple: 128 0 128

You can read more about RGB on the web, such as this link: https://www.rapidtables.com/web/color/RGB_Color.html.

Viewing PPM Images

PPM files are an uncommon file format, so if you double-click on a ".ppm" image file you will be unable to view it. These are a few options for viewing PPM image files, which will be important for testing and debugging.

1. Download a simple JavaScript program for viewing images on your local computer: ppmReader.zip (available on Blackboard). Download, double-click to open, and extract the file ppmReader.html. Save that .html file anywhere on your local computer. When you want to view a PPM image file, download the PPM file to your local computer, double-click on ppmReader.html, and select the PPM image file for viewing. This tool is also available online at https://www.cs.rhodes.edu/welshc/COMP141_F16/ppmReader.html.
2. On Windows you can view PPM images using Irfanview, a free image utility. If the installation fails, you may have to download the installer and run as administrator for it to install properly on Windows. To do so, right-click on the installer program and select "run as administrator". Here is the link to download Irfanview: https://www.irfanview.com/.
3. "View" PPM files in your local text editor. Go to the File menu, then the Open command, and then browse to the file and open it. You will see lots and lots of integers! If you don't see the PPM files listed in the Open File dialog, try selecting "All files" from the drop-down.

Implementation Details

You will need to create a way to map the colors to their RGB values, and you will need to finish the implementation of the geometry and screen interfaces. You will also need to finish writing main() to call the appropriate methods in order to match the output and fulfill the functionality of the program.

Color Map

The user will enter the color as a string, but this will need to be changed to the corresponding RGB value. As a result, you will need some way to map these two. **A Go map may be appropriate, but is not required. Feel free to change the type of Color from int to something else if you would like.** The colors recognized by the program are red, green, blue, yellow, orange, purple, brown, black, white (all lowercase).

Any other color entered should trigger the error in the program that indicates that the color is unknown. So, as you create your implementation for this color map, you may want to implement the function **`colorUnknown()`** which takes in a **`Color`** struct instance and returns **`true`** if the color is unknown and returns **`false`** if the color is known.

Here are the RGB values for the valid colors:

- red (255, 0, 0)
- green (0, 255, 0)
- blue (0, 0, 255)
- yellow (255, 255, 0)
- orange (255, 164, 0)
- purple (128, 0, 128)
- brown (165, 42, 42)
- black (0, 0, 0)
- white (255, 255, 255)

The `geometry` Interface

There are three types which implement the **`geometry`** interface:

- **`Rectangle`**, which consists of
  - **`ll`**: a **`Point`** representing the lower-left corner of the rectangle
  - **`ur`**: a **`Point`** representing the upper-right corner of the rectangle
  - **`c`**: the **`Color`** of the rectangle
- **`Triangle`**, which consists of
  - **`pt0`**: a **`Point`** representing the first point of the triangle
  - **`pt1`**: a **`Point`** representing the second point of the triangle
  - **`pt2`**: a **`Point`** representing the third point of the triangle
  - **`c`**: the **`Color`** of the triangle
- **`Circle`**, which consists of
  - **`center`**: a **`Point`** representing the center of the circle
  - **`r`**: an integer representing the radius of the circle
  - **`c`**: the **`Color`** of the circle

Note that **`Color`** above refers to the defined type in the program, and that **`Point`** is another struct which contains:

- **`x`**: the x position of the point
- **`y`**: the y position of the point

The **`geometry`** interface consists of the following methods:

- **`draw(scn screen) (err error)`**
  Purpose: Draw the filled-in shape on the screen.
  This method with a **`Triangle`** receiver has been fully implemented for you.
  Your job is to finish the implementation of this method with a **`Circle`** receiver and implement this method with a **`Rectangle`** receiver. If the user attempts to draw a shape that would go off screen, your program you should return the

`outOfBounds` error from this method, not draw anything on the screen, and print the corresponding error message.

- **`printShape() (s string)`**
  Purpose: Print the type of the object.
  Your job is to implement this method for all three receivers: **`Rectangle`**, **`Triangle`**, and **`Circle`**. Below are sample outputs (the return value of this method was used to print the outputs shown):

```
Rectangle: (100,200) to (300,400)
Triangle: (100,100), (600,300), (875,850)
Circle: centered around (500,500) with radius 45
```

The **`screen`** Interface

There is one type which implements the **`screen`** interface:

- **`Display`**, which consists of
  - **`maxX`**: the dimension of the display on the X-axis
  - **`maxY`**: the dimension of the display on the Y-axis
  - **`matrix`**: a slice of slices in which each element represents the current **`Color`** on the display for the given pixel (Note that **`Color`** here refers to the defined type in the program.)

The **`screen`** interface consists of the following methods, and your job is to implement each of them:

- **`initialize(x, y int)`**
  Purpose: Initialize a screen of maxX times maxY. It will be called before any other method in this interface. It should also clear the screen so that it is all white.
- **`getMaxXY() (x, y int)`**
  Purpose: Retrieve and return the maxX and maxY dimensions of the screen.
- **`drawPixel(x, y int, c Color) (err error)`**
  Purpose: "Draw" the pixel with a given color at a given location. By "draw", the value of the matrix for that particular pixel should be updated with the appropriate color. If the pixel X and Y values given are out of bounds, or if the color is unknown, return the appropriate error.
- **`getPixel(x, y int) (c Color, err error)`**
  Purpose: Retrieve and return the color of the pixel at a given location. If the color is unknown, return the appropriate error.
- **`clearScreen()`**
  Purpose: Clear the whole screen by setting each pixel to white.
- **`screenShot(f string) (err error)`**
  Purpose: Write the screen to a .ppm file. Use the matrix and your color map to write the corresponding information to a PPM file as described earlier (which you

can then view graphically with an image viewer). If there is some error creating a PPM file with the name specified, print the error.

The Main Program

You will need to finish implementing **`main.go`** so that it uses the interfaces that were created.

The program starts by asking the user for the dimensions of the display. It then runs a loop that allows the user to draw shapes, until they enter "X" to stop drawing. Finally, it asks the user to enter the name of the file in which the resulting figure should be saved, and writes the pixel information and the header to a PPM file, before exiting the program.

You may assume that user input follows the format that is expected, i.e. the only errors that your program needs to detect and display are those for out of bound figures, unknown colors, and/or failure to create a PPM file with the name given.

See the sample output later in this document for what this flow of execution should look like.

Sample Output

Below is some sample output for the program. User input is shown in **`red`**.

```
Project 5: Geometry Using Go Interfaces
CS 341, Spring 2025

This application allows you to draw various shapes
of different colors via interfaces in Go.

Enter the width (x) that you would like the display to be: 1024
Enter the height (y) that you would like the display to be: 1024

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> R
Enter the X and Y values of the lower left corner of the rectangle: -1 12
Enter the X and Y values of the upper right corner of the rectangle: 100 200
Enter the color of the rectangle: red
Rectangle: (-1,12) to (100,200)
**Error: Attempt to draw a figure out of bounds of the screen.

Select a shape to draw:
         R for a rectangle
         T for a triangle
```

```
        C for a circle
 or X to stop drawing shapes.
Your choice --> R
Enter the X and Y values of the lower left corner of the rectangle: 200 200
Enter the X and Y values of the upper right corner of the rectangle: 400 400
Enter the color of the rectangle: purple
Rectangle: (200,200) to (400,400)
Rectangle drawn successfully.

Select a shape to draw:
        R for a rectangle
        T for a triangle
        C for a circle
 or X to stop drawing shapes.
Your choice --> R
Enter the X and Y values of the lower left corner of the rectangle: 1000 1000
Enter the X and Y values of the upper right corner of the rectangle: 1020 1020
Enter the color of the rectangle: magenta
Rectangle: (1000,1000) to (1020,1020)
**Error: Attempt to use an invalid color.

Select a shape to draw:
        R for a rectangle
        T for a triangle
        C for a circle
 or X to stop drawing shapes.
Your choice --> R
Enter the X and Y values of the lower left corner of the rectangle: 1000 1000
Enter the X and Y values of the upper right corner of the rectangle: 1020 1020
Enter the color of the rectangle: blue
Rectangle: (1000,1000) to (1020,1020)
Rectangle drawn successfully.

Select a shape to draw:
        R for a rectangle
        T for a triangle
        C for a circle
 or X to stop drawing shapes.
Your choice --> T
Enter the X and Y values of the first point of the triangle: 150 150
Enter the X and Y values of the second point of the triangle: 100 90
Enter the X and Y values of the third point of the triangle: 32 45
Enter the color of the triangle: black
Triangle: (150,150), (100,90), (32,45)
Triangle drawn successfully.

Select a shape to draw:
        R for a rectangle
        T for a triangle
```

CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

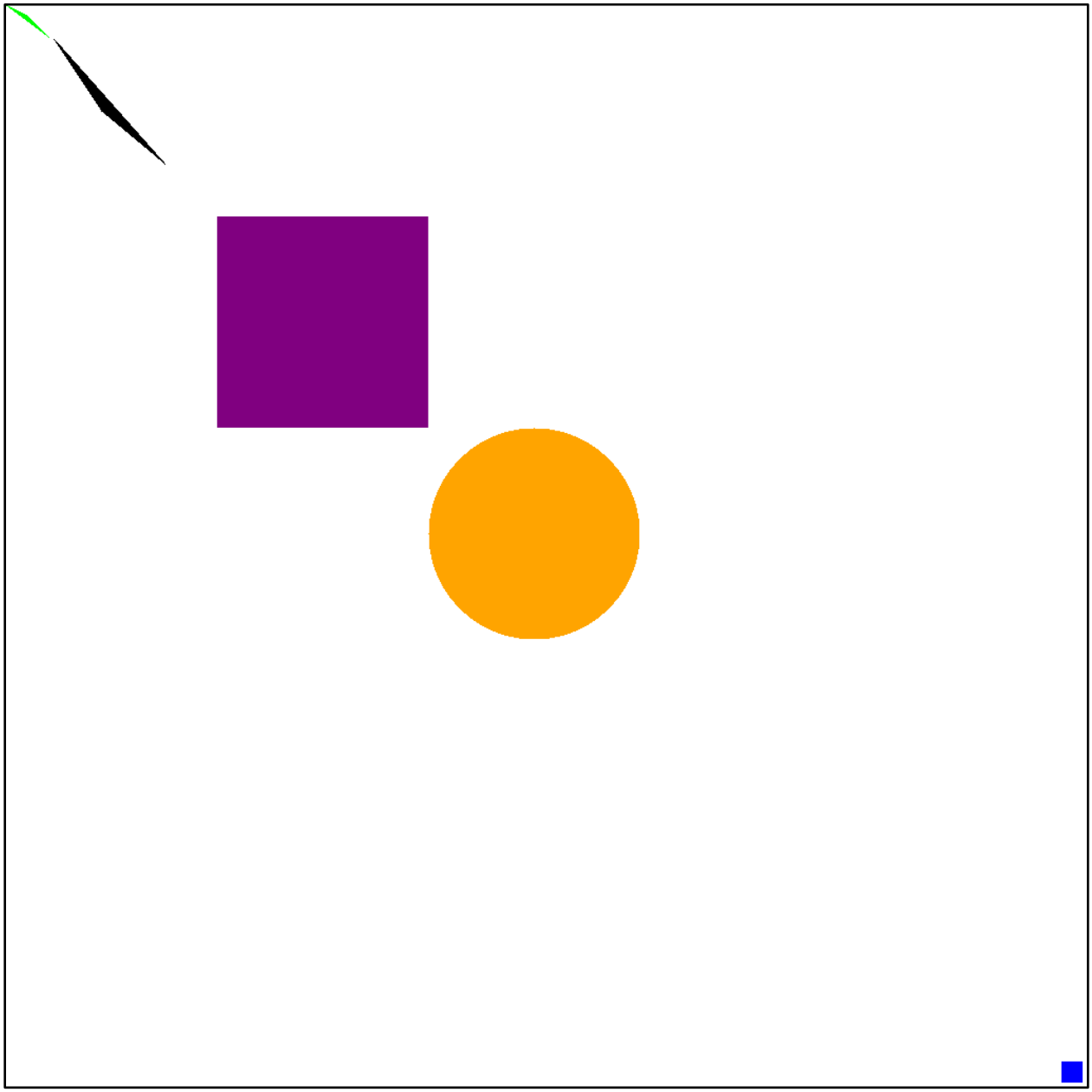
```
         C for a circle
 or X to stop drawing shapes.
Your choice --> T
Enter the X and Y values of the first point of the triangle: 0 0
Enter the X and Y values of the second point of the triangle: 10 20
Enter the X and Y values of the third point of the triangle: 30 40
Enter the color of the triangle: green
Triangle: (0,0), (10,20), (30,40)
Triangle drawn successfully.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> T
Enter the X and Y values of the first point of the triangle: 0 1050
Enter the X and Y values of the second point of the triangle: 900 950
Enter the X and Y values of the third point of the triangle: 42 500
Enter the color of the triangle: chartreuse
Triangle: (0,1050), (900,950), (42,500)
**Error: Attempt to draw a figure out of bounds of the screen.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> T
Enter the X and Y values of the first point of the triangle: 0 1020
Enter the X and Y values of the second point of the triangle: 900 950
Enter the X and Y values of the third point of the triangle: 42 500
Enter the color of the triangle: chartreuse
Triangle: (0,1020), (900,950), (42,500)
**Error: Attempt to use an invalid color.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> C
Enter the X and Y values of the center of the circle: 500 500
Enter the value of the radius of the circle: 100
Enter the color of the circle: orange
Circle: centered around (500,500) with radius 100
Circle drawn successfully.

Select a shape to draw:
```

CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

```
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> C
Enter the X and Y values of the center of the circle: 1024 1024
Enter the value of the radius of the circle: 10
Enter the color of the circle: gray
Circle: centered around (1024,1024) with radius 10
**Error: Attempt to draw a figure out of bounds of the screen.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> C
Enter the X and Y values of the center of the circle: 1014 1014
Enter the value of the radius of the circle: 10
Enter the color of the circle: gray
Circle: centered around (1014,1014) with radius 10
**Error: Attempt to draw a figure out of bounds of the screen.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> C
Enter the X and Y values of the center of the circle: 1010 1010
Enter the value of the radius of the circle: 10
Enter the color of the circle: gray
Circle: centered around (1010,1010) with radius 10
**Error: Attempt to use an invalid color.

Select a shape to draw:
         R for a rectangle
         T for a triangle
         C for a circle
 or X to stop drawing shapes.
Your choice --> X
Enter the name of the .ppm file in which the results should be saved: sample
Done. Exiting program...
```

CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

The resulting file is called **`sample.ppm`**. Opening the PPM file in a viewer yields the following image (border added to make size of image and relative size of shapes clearer):



CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

Running the Program

Once you have downloaded the files, and you have put them into the folder that you would like to work in, initialize the Go module by navigating to the folder in your terminal and then typing **`go mod init main`** into the terminal.

Then you should be able to run the program using the command **`go run .`** in your terminal.

Feel free to ask any of the instructional staff via drop-in hours and/or Piazza if you are having trouble with this.

Requirements

The structure of the program should consist of two files, **`main.go`** and **`draw.go`**, and is part of the **`main`** package. **You must implement the interfaces and methods as described above. Use only the packages `fmt`, `math`, `errors` and `os`.**

It is also expected that you use good programming practices, i.e. functions, comments, consistent spacing, etc. In a 3xx class this should be obvious and not require further explanation. But to be clear, if you submit a program with no functions and no comments, you will be significantly penalized --- in fact you can expect a score of 0, even if the program produces the correct results. How many functions? How many comments? You can decide. Make reasonable decisions and you will not be penalized.

Submission

Login to Gradescope.com and look for the assignment “Project 05”.

Submit your **`main.go`** and **`draw.go`** files under “Project 05”.

To encourage local testing, you will be limited to 30 submissions (submissions that result in a score of zero do not count towards this limit).

Keep in mind that any manual grading for requirements, etc. will be based on your last submission unless you select an earlier submission for grading. If you do choose to activate an earlier submission, you must do so before the deadline.

After the project is due, the TAs will manually review the programs for style and adherence to requirements (0-100%). **Failure to meet a requirement will trigger a large deduction.**

No late submissions will be accepted for this assignment.

CS 341, Spring 2025
Project 5 – Geometry Using Go Interfaces
Due: Sunday 5/4/2025 at 11:59pm

Academic Integrity

All work is to be done individually — group work is not allowed.

While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own, etc. The University's policy is available here: https://dos.uic.edu/community-standards/

In particular, note that **you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance**. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums (e.g. you cannot download answers from Chegg). Other examples of academic dishonesty include emailing your program to another student, sharing your screen so that another student may copy your work, copying-pasting code from the internet, working together in a group, and allowing a tutor, TA, or another individual to write an answer for you.

Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at the link above.