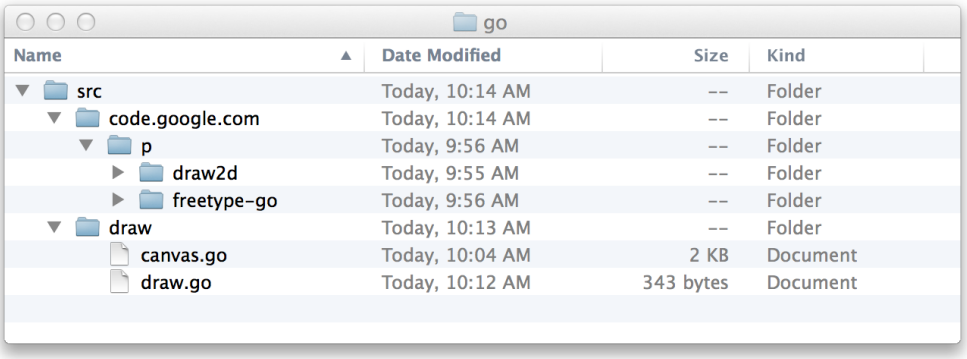
02-201 Homework 5: Graphics

Due: 11:59pm on Thursday, October 22

1. Set up

1. Create a directory called "`go`" someplace (different than where you have installed Go)
2. Inside of that directory create a directory called `src`
3. Download the template from Piazza, and unzip it into the `src` directory. You should now have a bunch of directories that look like this:



4. Set your GOPATH environment variable to the location of your `go` directory that you made above. On a Mac:

   ```
   export GOPATH=/Users/carlk/Desktop/go
   ```

   where you replace the directory name after the `=` with the location of the `go` directory you just made.

   On Windows use

   ```
   set GOPATH=C:\Users\carlk\Desktop\go
   ```

5. "Cd" into `go/src/draw/` and run `go build`. This should produce a file called `draw`. You can run that program by typing `./draw` (Mac) or `.\draw` (Windows). This will produce a picture file called `test.png`. Make sure this works and that file contains a picture of a triangle.
6. Edit `go/src/draw/draw.go` to include your solution to the assignment (described below).

2. Drawing using Canvases

2.1 Creating a new canvas

You can create a new image by using the following syntax:

```
canvas := CreateNewCanvas(w, h)
```

where `w` and `h` are the width and height of the image. This call returns:

- a canvas, of type `Canvas`, that we will be able to draw on.

For reference, the `CreateNewCanvas` function has the following signature:

```
func CreateNewCanvas(w, h int) Canvas
```

You can create as many canvases as you want — just call them something different:

```
c2 := CreateNewCanvas(200, 200)
```

When you create a new canvas, it will start as a blank white field. The current stroke and fill colors will be black (see below).

2.2 Drawing on the canvas

To draw on this canvas, you use the syntax:

```
canvas.FUNCTION(...)
```

where `canvas` is your canvas variable, and where `FUNCTION` is one of the following functions:

- `MoveTo(x, y float64)` — move the current point to position (x, y).
- `LineTo(x, y float64)` — logically draw a line from the current point to (x, y) and make (x, y) the current point using the current line color and current line width. Nothing is drawn until you call `Stroke()` or `Fill()` (see below).
- `SetStrokeColor(c color.Color)` — set the current line color to `c` (see below for colors).
- `SetLineWidth(width float64)` — set the current line width to `width`.
- `SetFillColor(c color.Color)` — set the current fill color to `c`.
- `Stroke()` — actually draw the lines specified by your `LineTo` calls, and clear the pending lines.
- `FillStroke()` — actually draw the lines specified by your `LineTo` calls (like `Stroke`) but in addition fill the region inside the lines with the current fill color.
- `Fill()` — like `FillStroke()` but only do the fill part (don't draw the lines).
- `Clear()` — fill the entire canvas with the current fill color.
- `ClearRect(x1, y1, x2, y2 int)` — fill the rectangle $(x1, y1) \times (x2, y2)$ will the current fill color.
- `Width()` — returns the width of the canvas (as an int)
- `Height()` — returns the height of the canvas (as an int)
- `SaveToPNG(filename string)` — write out the current picture to the given file.

2.3 Colors

To make a new color, use:

```
myColor := MakeColor(r,g,b)
```

where `r`, `g`, `b` are the amount of red, green, and blue in the color (each ranging from 0 to 255, where 0 means none of that color and 255 means the maximum amount of that color). For example:

```
myRed := MakeColor(255, 0, 0)
myBlue := MakeColor(0, 0, 255)
myGreen := MakeColor(0, 255, 0)
myPurple := MakeColor(128, 0, 128)
```

For reference, the `MakeColor` function has the following signature:

```
func MakeColor(r,g,b uint8) color.Color
```

You need to import `image/color` if you want to use the type `color.Color` directly.

2.4 Examples

2.4.1 Example 1: A rectangle

For example, to draw a long, skinny, red rectangle:

```
pic := CreateNewCanvas(1100, 300)
pic.SetStrokeColor(MakeColor(255, 0, 0))
pic.MoveTo(100,100)
pic.LineTo(100, 200)
pic.LineTo(1000, 200)
pic.LineTo(1000, 100)
pic.LineTo(100, 100)
pic.Stroke()
pic.SaveToPNG("MyRedRectangle.png")
```

This will produce a file called `MyRedRectagle.png` with the following picture:

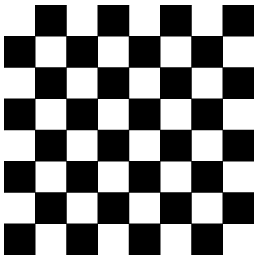
2.4.2 Example 2: A checkerboard

```
// draw a single square at (r,c) on the board b of width w, h
func drawCheckerSquare(b Canvas, r, c int, h, w float64) {
    x1, y1 := float64(r) * h, float64(c) * w
    x2, y2 := float64(r+1) * h, float64(c+1) * w
    b.MoveTo(x1, y1)
    b.LineTo(x1, y2)
    b.LineTo(x2, y2)
    b.LineTo(x2, y1)
    b.LineTo(x1, y1)
    b.FillStroke()
}

func drawCheckerBoard(w, h int) {
    board := CreateNewCanvas(w, h)
    squareWidth := float64(w) / 8.0
    squareHeight := float64(h) / 8.0
    board.SetFillColor(MakeColor(0,0,0))

    for r := 0; r < 8; r++ {
        for c := 0; c < 8; c++ {
            if (r % 2 == 0 && c % 2 == 1) || (r % 2 == 1 && c % 2 == 0) {
                drawCheckerSquare(board, r, c, squareHeight, squareWidth)
            }
        }
    }
    board.SaveToPNG("CheckerBoard.png")
}
```

This will produce a file called `CheckerBoard.png` with the following picture:



Notes:

- black is the color `MakeColor(0,0,0)` and white is the color `MakeColor(255,255,255)`. (Remember that white light is the combination of all colors.)
- You can pass a canvas to a function just as with any other kind of variable. The type of a canvas is `Canvas`.

3. Assignment

In this assignment, you will write a program to draw some of the things you've computed in previous assignments. There are 3 parts.

Your program should be able to be run using the following command line:

```
draw r d RULE
```

where

- `r` is a real number > 0.
- `d` is a real number > 0.
- `RULE` is CA rule as specified in assignment 3. (I.e. a string of 8 0s and 1s.)

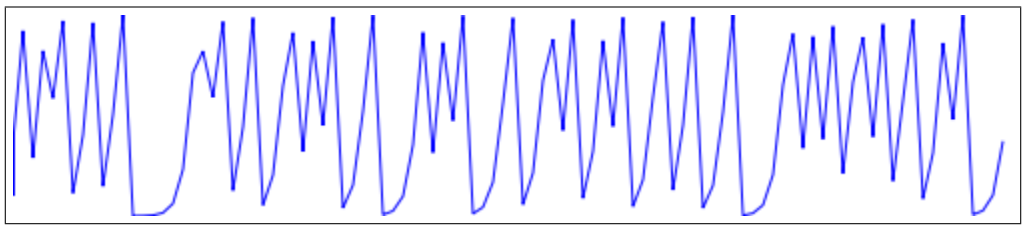
3.1 Part 1: Draw the Pop Size function

Recall the `PopSize(r, x0, max_t)` function from Homework 1.

Create a 500 by 100 canvas, and draw a line that connects the points $(5 \times t, 100 - 100 \times x(t))$, for $t = 0 \ldots 99$, where $x(t)$ is the population size at time t when you assume `x_0` $= 0.1$ and `max_t` $= 100$. The parameter r for your pop size function should be taken from the `r` parameter in the command line above.

Your plot should use a blue line of width 1.

In other words, you should end up with a plot that looks like this (when $r = 4$ for example):



You should save the image to a file called `PopSize.png`.

3.2 Part 2: Draw your random walk

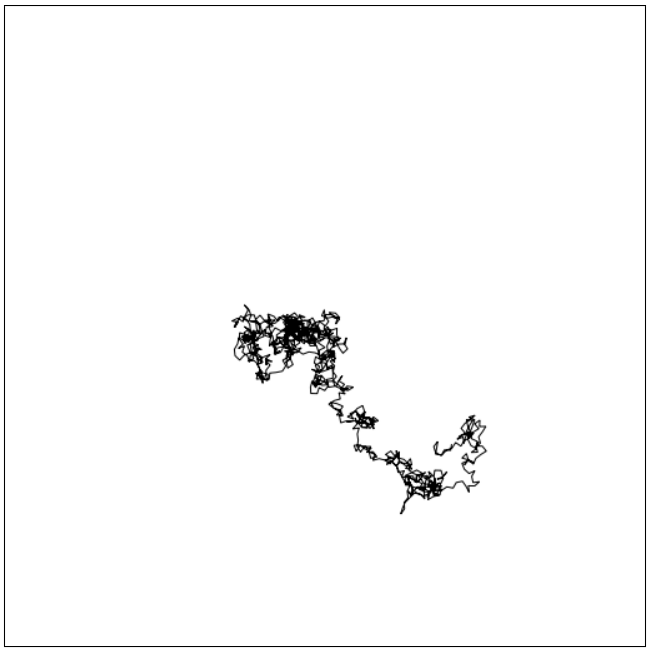
Create a 500 by 500 canvas, and draw a line connecting the points that your random walk from assignment 2 would have visited assuming that program was run via:

```
go run randwalk.go 500 500 d 1000 12345
```

That is, your random walk field is 500 by 500, the step size `d` is given by the `draw` command line above, your walk runs for 1000 steps, and the random seed is always 12345.

You should draw using a black line of width 1.

In other words, you should end up with a picture that looks like this:



You should save the image to a file called `RandomWalk.png`.

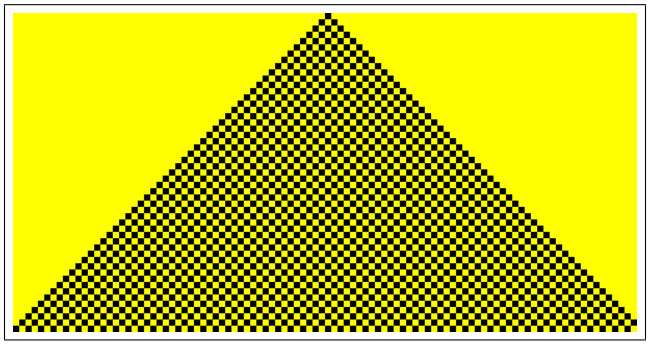
3.3 Part 3: Draw the output of a cellular automata

Create a 500 by 255 canvas. Consider the output of your CA program from assignment 3, assuming it was run using the command line:

```
go run ca.go RULE 100 50
```

Draw the output of your cellular automata where each 5 pixel band is one line of your previous output lines and where `#` is drawn as a 5-by-5 black square and a space as a 5-by-5 yellow square.

In other words, you should end up with a picture that looks like this (when RULE=11111010):



You should save the image to a file called `CA.png`.

3.4 Tips on how to start

First, install the template provided in the assignment on BlackBoard and make sure you can:

```
go build
./draw
```

(On Windows you may have to type `.\draw`.) This ensures you are set up to begin coding. **Do this today.**

Next, copy and paste the relevant functions from your past assignments (you can download them from auto lab if you don't have them) into the `draw.go` file. Make sure you can still run `go build` without any errors; fix any errors that occur.

Next, write the code to read the command line parameters. Use your code from assignments 2 and 3 for a guide.

Next, modify your previous functions to create a canvas, and draw on it as needed, and to save it to a file.

Focus on just one picture at a time. Get Part 1 done, and then move on to Part 2.

You can assume the user provides correct input on the command line.

3.5 Learning outcomes

After completing this assignment, you should

- be able to use a drawing package,
- have gained an understanding about code reuse,
- be able to compile your programs to an executable file,
- gained additional practice using functions, variables, loops, and if statements.

3.6 Extra Credit!

Have your program also output an image called `MyCoolPicture.png`, that is a drawing of some cool picture of your design.