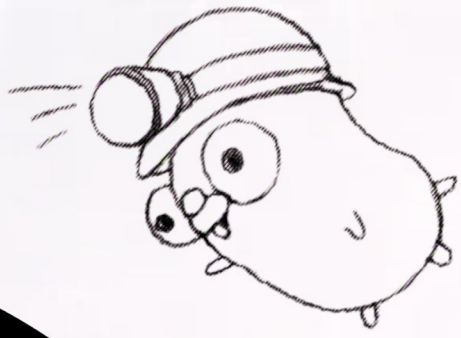


A CREATIVE GUIDE

GENERATIVE ART IN GO

by Preslav Rachev

Generative Art in Go

A creative guide

Preslav Rachev

This book is available at https://leanpub.com/generative-art-in-golang

This version was published on 2026-06-18

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

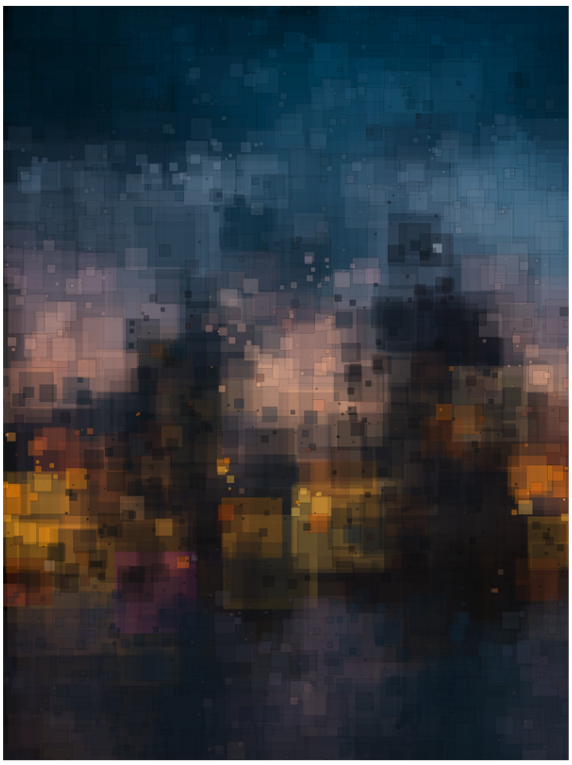
Contents

Acknowledgments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

1. Introduction

Welcome to the wonderful world of using Go to push pixels on screen, or as we will do in this tutorial, inside a bitmap image.



1.1 My Story

I have never been particularly good at drawing, but at a very early age, I discovered the power that a simple algorithm can have when applied multiple times over. Like many kids of the late 80s and 90s, I started programming using BASIC, PASCAL, and C later on. All I wanted to do was develop games and all sorts of interactive stuff. Those were fun times.

As I was growing up, reason told me to think of my newly discovered skill of programming in a more business sense. Games gave way to forms and spreadsheets. Naturally, in Undergraduate school, I chose a Business degree to accompany my foray into computer science.

It wasn't until I came to Germany to pursue my Master's degree that my passion for digital art rekindled. I had a chance to be a student of the great Frieder Nake[1] while doing my Master's degree. That was how I discovered Generative Art and the Processing[2] language.

Some of my early Processing work - 2010-2012

After graduating from my Master's, my reason prevailed once again. I came back to the well-beaten path of Web and Enterprise application software development, but this time, I kept my artistic passion closer.

About a couple of years ago, I picked up the Go programming language, primarily due to its natural fit for my day job - easy development of CLI tooling, HTTP servers, and infrastructure. I had an instant flashback to the days when I programmed in C as a kid. I decided to prove to myself and others that this tiny language can deliver more than what most programmers get out of it. Naturally, I started pushing pixels with it, and this is how this guide was born.

1.2 What did you just pay for?

This guide is not a programming language course in the traditional sense. It will not teach you the Go programming language for two reasons. First, there are hundreds of outstanding books and guides out there. Second, one of the core characteristics of the language itself is that it is easy to learn. Even a programmer without much prior experience can skim through *Go by Example*[3] and get a basic idea of how the language works in no time. If you want to get up to speed with the language, I can highly suggest going through the Tour[4] first and then checking out the interactive tutorials over at Play with Go[5].

[1] https://en.wikipedia.org/wiki/Frieder_Nake
[2] https://processing.org/
[3] https://gobyexample.com/
[4] https://tour.golang.com/list
[5] https://play-with-go.dev/

This guide won't teach you how to create art either. Nobody can. As we all know too well, **beauty is in the eye of the beholder**. If you like what comes out of your code, it's your art, and no one will have the right to dispute it.

The idea of this guide is to give you a solid foundation by showing you one specific approach (the one I called *Squares*) and then leave the rest to your imagination.

1.3 Who is this for?

This tutorial is for people of all kinds and levels of programming experience. It is highly recommended that you know a bit of Go, but then again, unlike other programming languages, picking it up will not take you long. Since things in this tutorial are pretty basic, it can serve as a perfect companion to those new to the Go language.

At the same time, I hope to be able to show experienced programmers a new side of the language. One that has little to do with tooling, servers, or infrastructure, but one that might prove to be just as valuable in potential future projects.

Let's start, shall we?

1.4 Chapter Overview

Before we move on, here is what to expect from the following few chapters, in brief:

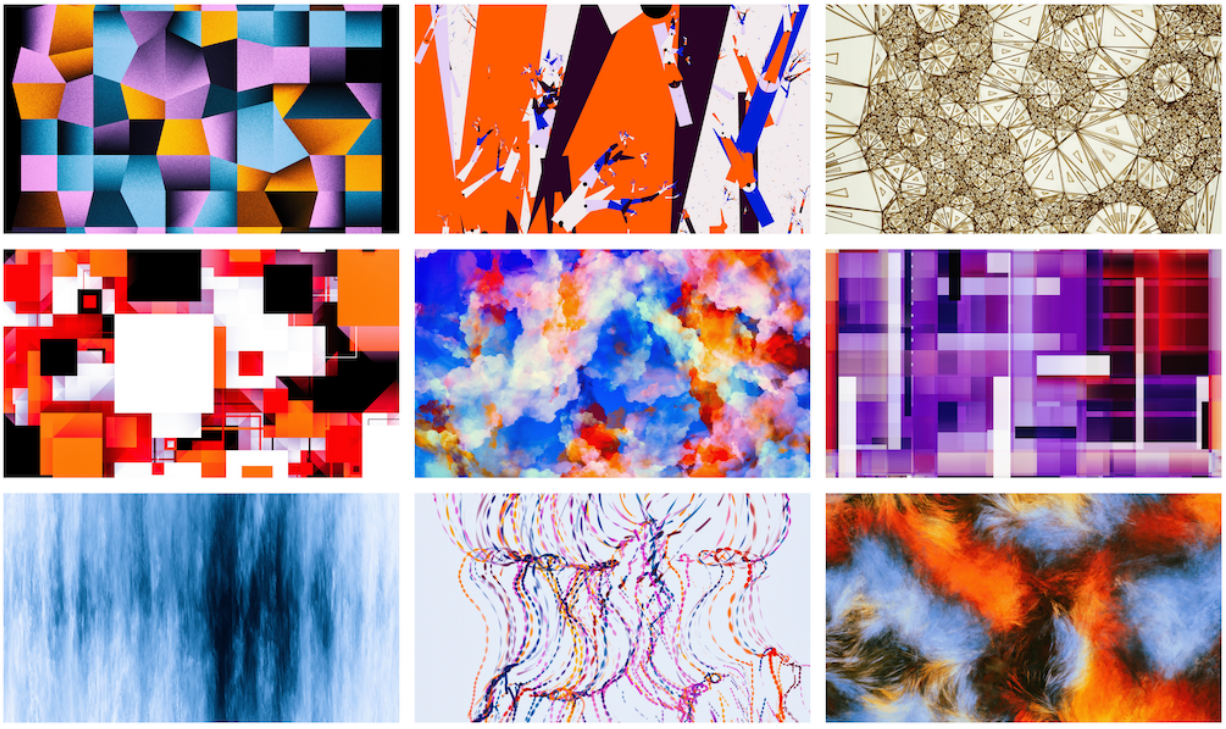
- **Chapter 2** will introduce the topic of algorithmic art, its origins and basic concepts.
- **Chapter 3** will explain the basic building blocks of Go's standard library. The reader will learn the difference between the core `image` primitives, how to work with colors, as well as how to do basic image manipulation using the standard library alone.
- **Chapter 4** is where the fun part begins. This chapter will show the reader how to set up a project from scratch, and explain step-by-step how to implement sketches similar to the ones shown at the beginning of this chapter.
- **Chapter 5 (Bonus)**: To give the reader even more practical value, this chapter will take the simple sketch implemented in **Chapter 4** and turn it into an end-user Web application. The application will allow anyone to generate their own art pieces directly in the Web browser, using a cutting-edge technology called WebAssembly (WASM)[6] .

  Although I would be happy to see everyone reading this tutorial from end to end, if all you came here for is the source code of the sketch, you can find it right at the end of **Chapter 4**.

NOTE: All the companion source code to this book is available on GitHub[7]. Readers are more than encouraged to fork the repository and submit pull requests with suggestions for further improvements.

[6] WebAssembly
[7] https://github.com/preslavrachev/generative-art-in-go

2. Generative Art: Definition and Origin



Selected artworks by Argentinian artist Manolo Gamboa Naon. All chosen artworks published with the author's permission.

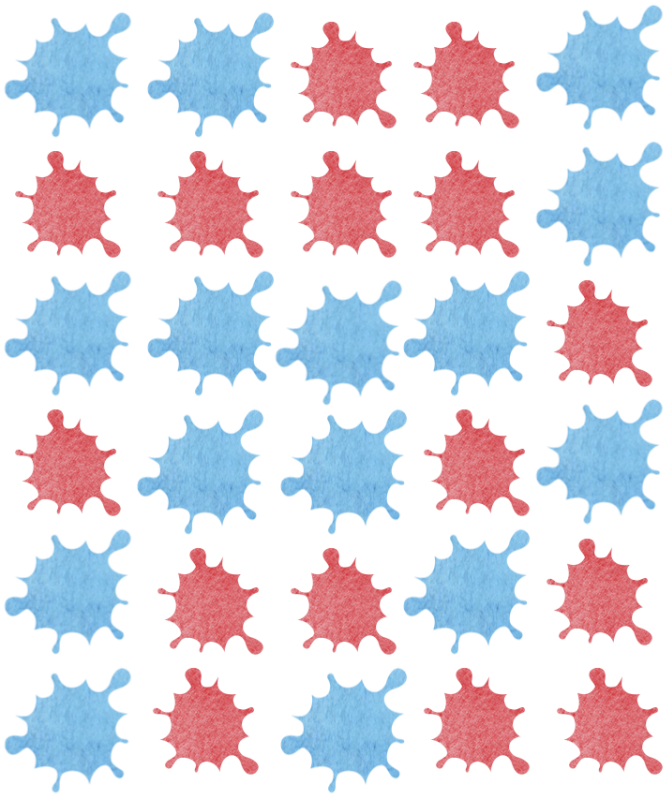
[1]

Before we get our hands into writing code, let us define what we will be working on, shall we? My goal with this guide is not only to give you the raw code, but also, to inspire many of you to learn more about the principles and background of algorithmic art.

Suppose, we have a dice, a plain canvas, a brush, and two buckets of paint - red and blue. Starting from the top left corner down to the bottom right one, we will fill the canvas with tiny paint splatters. The outcome of the dice will help us decide the color of every next splatter. If it is an even number, we will use the red colour, otherwise, we will use blue.

Here is what I've got:

[1] http://manoloide.com/



Congratulations, we have just created our first algorithmic art piece! Not super pretty, but is going to help us define what we are going to be dealing with.

Generative art is

1. a result of applying a *finite* number of steps over and over again
2. a result of the fine balance between human control and total entropy (randomness)

In his 2003 paper, artist and scholar Philip Galanter defines Generative Art as "*any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art*"[2]. Such an autonomous system usually starts with some human guidance (initial setup), but then goes on its own, taking one decision after another, until it reaches its final destination. The fact that it is the system, and not the individual who produces the final output is a crucial characteristic of this type of artistic expression.

Refer to our simple example above. We have our initial configuration (white canvas, blue and red paint) and basic rules (use red when the dice's outcome is even, otherwise, blue). Our system took the form of subsequent steps. Each step took into account our initial configuration, as well as the current state (place the new splatter right next to the previous one, move to a new line, etc).

[2] https://www.philipgalanter.com/downloads/ga2003_paper.pdf

To make our piece generative, we added one more element - a *level of uncertainty*. This is the "secret sauce" of most, if not all generative artworks. In our example, the dice played the role of Destiny, choosing the color of every next splatter. In fact, randomness, plays such a big role in generative artwork that I could pay tribute and dedicate an entire chapter to it alone. In most cases, generating random numbers is enough, but it is worth pointing out that many artworks draw uncertainty from other sources too - viewer interaction, real-time statistics, bio- or chemical reactions, etc.

In the example above, the rules are too simple, therefore the final result looks *noisy*. If on the other hand, we had made the rules such that randomness had little or no effect, the output would look too *structured*. Achieving the so called aesthetic *emergence* is the result of creating rules, such that, at a certain point, make the human brain stop thinking about the outcome as random noise, but starts searching for familiar patterns into it.

2.1 History

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.1.1 Generative Art and early scientific foray into AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.2 Processing and the new generation of artists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.2.1 OpenProcessing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.3 Other tools worth mentioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.3.1 openFrameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

2.3.2 NodeBox

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3. Go and Graphics Programming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.1 Let's Go!

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.1.1 My story with Go

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.2 Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.3 Graphics in Go

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.3.1 Core Packages and Types in the Go Standard Library

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.3.2 Vector vs. Raster Graphics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.3.3 Basic Image Types

At the core, there are two main interfaces: `image.Image` and `image/draw.Image`. One serves as a readable image source, the other is used for drawing:

src/image/image.go

```
// image.Image
type Image interface {
    ColorModel() color.Model
    Bounds() Rectangle
    At(x, y int) color.Color
}
```

src/image/draw/draw.go

```
// draw.Image is an image.Image with a Set method to change a single pixel.
type Image interface {
    image.Image
    Set(x, y int, c color.Color)
}
```

Note that both `image.Image` and `image/draw.Image` are **interfaces** and not **structs**. This is a design decision that allows infinite possibilities for modifying an existing image, without changing its consumers. Those of you who have worked with Go long enough will note a similarity between `image.Image` and `io.Reader` as well as between `image/draw.Image` and `io.Writer` . Conceptually, the idea is the same - provide a simple high level API for consuming and modifying an image, by hiding what is happening under the hood.

3.3.4 What is Bounds() used for?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4 Color

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4.1 What is in a color?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4.2 The image/color.Color interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4.3 RGBA and NRGBA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4.3.1 It's about those `uint`s

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.4.3.2 Key takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.5 Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.5.1 Ex 1: Circular Mask

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.5.2 Ex 2: Grayscale Filter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.5.3 Ex 3: Image Resizing Using Nearest Neighbors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.6 Additional Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.6.1 fogleman/gg

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

3.6.2 nfnt/resize and disintegration/imaging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4. Sketch Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.1 The Algorithm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.2 Creating the Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.2.1 $GOPATH and Go Modules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.2.2 Creating a new module for our project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.3 The Sketch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.3.1 Starting Configuration

Having our `sketch` package declared, let's create a type representing our user-defined parameters:

```go
package sketch

// imports ...

type UserParams struct {
        DestWidth               int
        DestHeight              int
        StrokeRatio             float64
        StrokeReduction         float64
        StrokeJitter            int
        StrokeInversionThreshold float64
        InitialAlpha            float64
        AlphaIncrease           float64
        MinEdgeCount            int
        MaxEdgeCount            int
}
```

These will be the starting set of "rules" we provide to the algorithm. By changing one or more of them, we can achieve significantly different results. Let's give a brief explanation to each parameter:

- `DestWidth` and `DestHeight` are the dimensions of our end sketch. Regardless of how large or small our source image is, we can produce a sketch of arbitrary dimensions, as long as our computers have enough capacity to render it.
- `StrokeRatio` - determines the size of the initial stroke compared to that of the final result. Our idea is to start big and reduce the stroke size upon each iteration. This is why an initial `StrokeRatio` of 3/4, or 0.75 should be a reasonable default value. Don't worry, we will have a separate code block with all default values I have found to work well.
- `StrokeReduction` - directly related to `StrokeRatio`. The reduction is the step, with which the initial stroke size gets minimized upon each loop iteration. I found that 0.002 works well in this case
- `StrokeJitter` - Deviation of our colored stroke from its projected position in the original image. If you recall the algorithm, a color gets picked at certain coordinates (X, Y) in the original image an a polygon with that color gets "drawn" onto the destination. The *jitter* helps to make the final image more unpredictable. By adding an element of randomness, the *jitter* would *shake* the hand of our algorithm, so that the stroke ends *near* where it should have been in the original image, but not exactly. I have computed this value to be a certain portion of the output image's width: `int(0.1 * float64(destWidth))`
- `StrokeInversionThreshold` - determines the minimum stroke size, beyond which we start adding borders around the stroke, for more contrast. To deepen the contrast even more. borders will be the negative of the brightness of the stroke's color. A bright stroke will get a dark border, and vice versa. To make and effect of depth-of-field in my sketches, I keep this value low - 0.05, so that the algorithm starts adding borders only when the stroke is at 5% of its original size, but not before.

- `InitialAlpha` - controls the stroke transparency at the start. To achieve the desired effect, stroke transparency is low at the beginning and increases at every step. Alpha is a floating point number that varies between 0 and 255. We can set it to a pretty low value: 0.1
- `AlphaIncrease` - is the step of transparency increase at each iteration. This one may differ from setup to setup, so feel free to play with it. In this case, I have set it to 0.06.
- `MinEdgeCount` and `MaxEdgeCount` - the minimum and maximum number of edges of each stroke, respectively. Each stroke will be a n-edge polygon. The defaults here I leave totally to the imagination of the reader. If you prefer simpler shapes, you can play with drawing just triangles and squares (3, and 4, respectively), or go for more complex shapes.

Let's add all the default values. Open the `cmd/main.go` file and write the following:

cmd/main.go

```
1  package main
2
3  // imports
4
5
6  func main() {
7          params := sketch.UserParams{
8                  DestWidth:                destWidth,
9                  DestHeight:               2000,
10                 StrokeRatio:              0.75,
11                 StrokeReduction:          0.002,
12                 StrokeInversionThreshold: 0.05,
13                 StrokeJitter:             int(0.1 * float64(destWidth)),
14                 InitialAlpha:             0.1,
15                 AlphaIncrease:            0.06,
16                 MinEdgeCount:              3
17                 MaxEdgeCount:              4
18         }
19 }
```

4.3.2 The Sketch Type

4.4 Load a source image from a file

4.4.1 I got an error right away. The image was loaded, but my application complained about “unknown format”. What do I do?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.5 Save the output of an image to a file

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.6 Setting up the Sketch instance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.6.1 Q: Are random numbers really random?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.7 Implementing the `Update` method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.8 Taking our sketch for a spin

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

4.9 The final sketch code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5. Bonus Chapter: Turning our sketch into an interactive Web application using WebAssembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.1 Running Go code in the browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.1.1 How about compiling Go to JavaScript?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.1.2 WebAssembly to the rescue

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.1.3 Limitations of WebAssembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.1.4 Go and JavaScript / DOM interoperability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.2 Creating a scaffold for our new sketch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.2.1 Compiling a WASM binary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.3 Loading a WASM binary in a Web application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.3.1 A short intro to Web Workers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.3.2 Creating our Web Worker

Let's create a simple Web Worker for our needs.

NOTE: Since this is a book about Go and not about JavaScript, I will abstain from going into too many details about the workings of the JavaScript code itself. There are numerous books and other resources out there, which will do a much better than my best efforts to do so.

In the root folder of your project, create a file called `go_worker.js`. Open it, and let's start putting our code together. Let's begin with a quick helper:

```
1 // hijack the console.log function to capture stdout
2 console.log = (line) => {
3   postMessage({
4     log: line,
5   });
6 };
```

Recall that Web Workers do not have access to the scope of the main window and the main UI thread. Depending on the browser version, this may include the console too. Thankfully, we can resolve this issue by *posting a message* of a particular kind. When the main thread catches this type of message, it would pass the `line` to the regular `console.log`. We will implement the message handler part in the next section.

Let's move on to the first major block of our Web Worker - loading our WASM file:

go_worker.js

```
self.importScripts("wasm_exec.js");

const go = new self.Go();

let mod, inst;
let result;

WebAssembly.instantiateStreaming(fetch("main.wasm"), go.importObject)
  .then((result) => {
    mod = result.module;
    inst = result.instance;

    go.run(inst);
    console.log("WASM binary Loaded");
  })
  .catch((err) => {
    console.error(err);
  });
```

`importScripts`[1] *synchronously* imports one or more scripts into the worker's scope. There are two things to note here. First, unlike the bare-bones scaffolding that we copied from `GOROOT`, in practice, when using a Web Worker, it is the worker itself that loads the `wasm_exec.js` bridge and not the main thread. Second, note the use of `self` instead of `window`. Inside a Web Worker, `self` would refer to the scope of the worker instance itself (unlike `this`, which changes its reference based on the current block of execution).

The rest of the code will load and instantiate our WASM binary.

5.3.3 Loading the Web Worker from the main application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.3.4 Serving and taking our setup for a test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

[1] https://developer.mozilla.org/en-US/docs/Web/API/WorkerGlobalScope/importScripts

5.4 Re-implementing the sketch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.4.1 Loading and sending a source image to the WASM application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

5.4.2 Rendering the output image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.

Special Thanks and Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generative-art-in-golang.