Ever wonder how it was generated? ***The image depicts 100 simulations of an asymmetric random walk.*** In this post, I'll go through the code used to generate this image. All the code can also be found here.

For $t = 1, 2, \ldots$, consider a series of i.i.d. random variables $X_1, X_2, \cdots$ such that for any t, $X_t = 1$ with probability p, and $X_t = -1$ with probability $1 - p$. ($p \in [0, 1]$ is a parameter that we get to choose.) A random walk simply tracks the cumulative sum of these random variables, i.e.

$$S_0 = 0, \quad S_t = \sum_{s=1}^{t} X_s.$$

In my image, I let the random walk run until it hits a fixed upper limit or a fixed lower limit. Here is an R function that generates one realization of this random walk:

```
# returns the random walk path values as a vector
# (random walk always starts at 0)
# p: probability of increasing by 1
# stop if path value hits either `lower` or `upper`
run <- function(p, lower, upper) {
    values <- c(0)
    current <- 0
    while (current > lower & current < upper) {
        current <- current + ifelse(runif(1) < p, 1, -1)
        values <- c(values, current)
    }
    values
}
```

(There might be more efficient ways of doing this, but since computation is relatively fast this is good enough for our purposes.)

The code below creates a list of 100 elements, each element being a simulated random walk. The probability of going up at any one step is 0.48, and we stop the random walk once we hit -50 or 50.

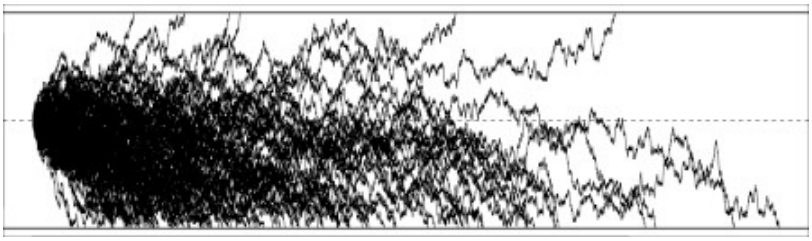
```
N <- 100  # no. of paths to simulate
p <- 0.48
lower <- -50
upper <- 50

# simulate paths
set.seed(1055)
vlist <- replicate(N, run(p, lower, upper))
```

We can plot these paths along with the x-axis and the upper & lower limits:

```
# get length of longest path
max_length <- max(sapply(vlist, length))

# make plot
par(mar = rep(0, 4))  # no margins
plot(c(1, max_length), c(lower, upper), type = "n")
for (i in 1:N) {
    lines(1:length(vlist[[i]]), vlist[[i]])
}
abline(h = 0, lty = "dashed")
abline(h = lower, lwd = 2)
abline(h = upper, lwd = 2)
```



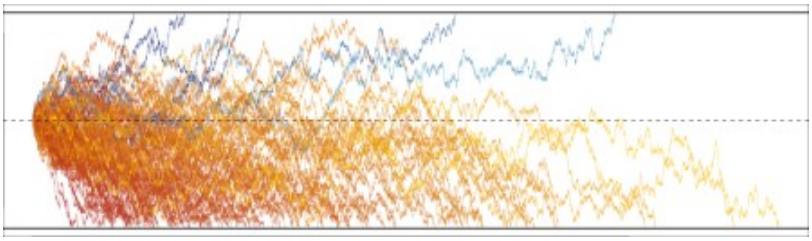
Without color, it's hard to make much sense of the image. To introduce color, let's create a function that picks a color for path based on (i) whether the path hit the upper or lower limit, and (ii) how long the path is when compared to the longest path in the image.

```
colorPicker <- function(values, max_length,
                        ls_color = c(178, 34, 34), ll_color = c(255, 204, 0),
                        us_color = c(0, 0, 102), ul_color = c(102, 204, 225)) {
    l <- length(values)
    if (values[l] < 0) {
        rgb_values <- (ls_color + (ll_color - ls_color) * l / max_length) / 255
    } else {
        rgb_values <- (us_color + (ul_color - us_color) * l / max_length) / 255
    }
    rgb(rgb_values[1], rgb_values[2], rgb_values[3])
}
```

If a path hits the lower limit and has length 0, it will have color `ls_color / 255`, and if a path hits the lower limit and has the longest length among all our simulated paths, it will have color `ll_color / 255`. There is a similar relationship for paths hitting the upper limit and `us_color` and `ul_color`. (You can see what these colors are at a website such as this.)

We can now color our black-and-white image:

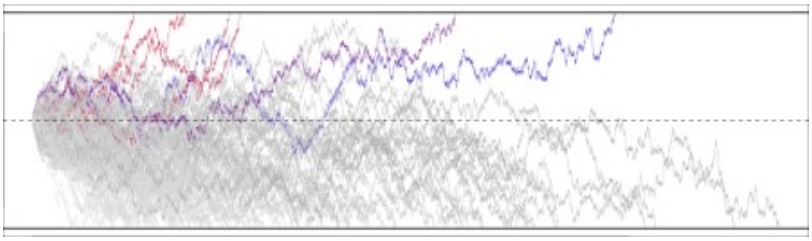
```
plot(c(1, max_length), c(lower, upper), type = "n")
for (i in 1:N) {
    lines(1:length(vlist[[i]]), vlist[[i]],
          col = colorPicker(vlist[[i]], max_length), lwd = 0.5)
}
abline(h = 0, lty = "dashed")
abline(h = lower, lwd = 2)
abline(h = upper, lwd = 2)
```



From the image it is now obvious that most of the paths hit the lower limit first, and fairly quickly at that! Here is the same image, but with different choices of color scale:

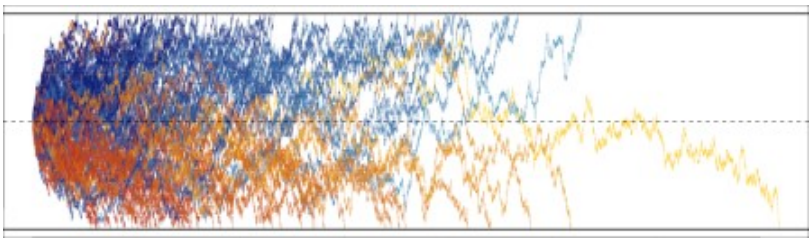
```
plot(c(1, max_length), c(lower, upper), type = "n")
for (i in 1:N) {
    lines(1:length(vlist[[i]]), vlist[[i]],
          col = colorPicker(vlist[[i]], max_length,
                            ls_color = c(230, 230, 230),
                            ll_color = c(166, 166, 166),
                            us_color = c(255, 0, 0),
                            ul_color = c(0, 0, 255)),
          lwd = 0.5)
```

```
}
abline(h = 0, lty = "dashed")
abline(h = lower, lwd = 2)
abline(h = upper, lwd = 2)
```

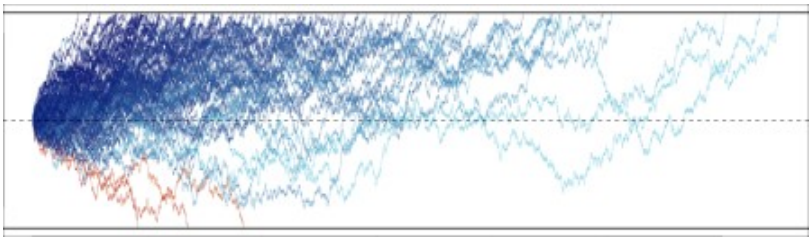


The possibilities are endless!

If you haven't seen random walks before, you might be surprised at how a slightly biased walk ($p = 0.48$ instead of $p = 0.5$) results in so many more paths hitting the lower limit before the upper limit, even though the two limits are the same distance from 0. This is an example of 100 simulations when $p = 0.5$:



This is an example of 100 simulations when $p = 0.52$:



Isn't it interesting how slight deviations in the probability of going one step up (instead of down) completely change the dynamics? It turns out that there is a closed form for the probability of hitting one limit (as opposed to hitting the other). If the upper limit is $x = b$ and the lower limit is $x = -a$, and $q = 1 - p$, then

$$\mathbb{P}\{\text{hit } x = b \text{ before } x = -a\} = \begin{cases} \frac{1-(q/p)^a}{1-(q/p)^{a+b}} & \text{if } p \neq 1/2, \\ \frac{a}{a+b} & \text{if } p = 1/2. \end{cases}$$

For my blog header image, $p = 0.48$, $a = b = 50$, which means the probability of hitting the upper limit is approximately 0.0179: not high at all!