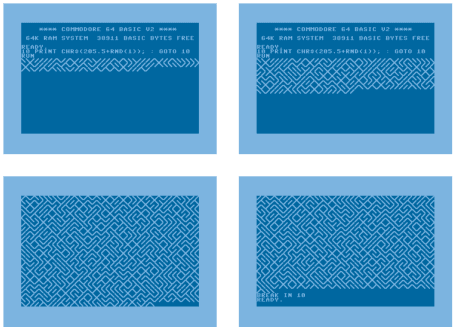
There is a celebrated Commodore 64 program that randomly prints outs / and \  
characters and fills the screen with neat-looking maze designs.

10 PRINT CHR$(205.5+RND(1)); : GOTO 10

Screenshots of the 10.

The basic idea, from my reading of the code, is that CHR$(205) is \,  
CHR$(206) is /, and the program randomly selects between the two by adding a  
random number to 205.5. Endlessly looping over this command fills the screen  
with that pleasing maze pattern.

In R, we could replicate this functionality with by randomly sampling the  
slashes:

sample\_n\_slashes <- function(n) {

sample(c("/", "\\"), size = n, replace = TRUE)

}

withr::with\_options(

list(width = 40),

cat(sample\_n\_slashes(800), sep = "", fill = TRUE)

)

#> //////\/\\/\/\/\/\\//\\\\//\//\/\//\///\

#> \\\/\//\\/\/////////////\\/\//\\\/\//\\\

#> /\\/\/\//\/\/\\/\\/\/\//\\\\//\/\\\/\///

#> \\//\/\\\/\///\\/\/\/////\//\///\/\/\//\

#> /\/\//\///\//\//\//\/\//\\/\\\\\/\\\//\/

#> //\\\\\//////\//\//\/\//\\////\/\\\/\/\/

#> \\////\/\\/\////\//////\\/\\/////\/////\

#> /\/\/\/\/\\///\/\\\\/\/\\//\/\\/\\\\\\//

#> //\\\/\///\///\/\\\//\//\\\\//\\/\\/\/\/

#> /\\/\//\\//\/\\\\\/\\\/\\\/\/\/\/\////\/

#> /\//\\//\////\\\///\//\/\\\/\\///\//\\\\

#> /\//\\////\/\\\//\\\//\/\///\\\\\/\/\\//

#> \\////\\/\\\\\\///\///\//\\\/\\\\//\////

#> \\\///\/\//\//\/\//\/\/\\\/\/\\///\/////

#> \\/\\//\\\\\\///\\\\/\\/\\//\//\\\/\\/\/

#> ////\\\////\/\\\\\/////\/\\\////\\///\\\

#> \//\\\//\///\/\\\\//\\\////\\//\\/\/\//\

#> \/\//\\//\\///\\\\\\//\///\/\\\\\\\\/\\/

#> ///\/\//\\/\/\\\\\\\\/\///\//\\///\\//\\

#> /////\\///\/\\/\/\\//\\//\/\\/\//\//\\\\

where withr::with\_options() lets us temporarily change the print width and  
cat() concatenates the slashes and prints out the characters as text.

We can also make this much prettier by drawing the patterns using ggplot2.

**Drawing line segments with ggplot2**

Instead of writing out slashes, we will draw a grid of diagonal line segments,  
some of which will be flipped at random. To draw a segment, we need a starting  
*x*–*y* coordinate and an ending *x*–*y* coordinate. geom\_segment() will  
connect the two coordinates with a line. Here’s a small example where we draw  
four “slashes”.

library(ggplot2)

library(dplyr)

data <- tibble::tribble(

~row, ~col, ~x\_start, ~x\_end, ~y\_start, ~y\_end,

1, 1, 0, 1, 0, 1,

1, 2, 1, 2, 1, 0, # flipped

2, 1, 0, 1, 1, 2,

2, 2, 1, 2, 1, 2)

ggplot(data) +

aes(x = x\_start, xend = x\_end, y = y\_start, yend = y\_end) +

geom\_segment()

The programming task now is to make giant grid of these slashes. Let’s start  
with an observation: To draw two slashes, we needed three *x*  
values: 0, 1, 2. The first two served as segment starts and the last two  
as segment ends. The same idea applies to the *y* values. We can generate a  
bunch of starts and ends by taking a sequence of steps and removing the first  
and last elements.

# We want a 20 by 20 grid

rows <- 20

cols <- 20

x\_points <- seq(0, 1, length.out = cols + 1)

x\_starts <- head(x\_points, -1)

x\_ends <- tail(x\_points, -1)

y\_points <- seq(0, 1, length.out = rows + 1)

y\_starts <- head(y\_points, -1)

y\_ends <- tail(y\_points, -1)

Each x\_starts–x\_ends pair is a column in the grid, and each  
y\_starts–y\_ends is a row in the grid. To make a slash at each  
row–column combination, we have to map out all the combinations of the rows  
and columns. We can do this with crossing() which creates all *crossed*  
combinations of values. (If it helps, you might think of *crossed* like [crossed  
experiments](https://en.wikipedia.org/wiki/Factorial_experiment) or the  
[Cartesian cross product](https://en.wikipedia.org/wiki/Cartesian_product) of  
sets.)

grid <- tidyr::crossing(

# columns

data\_frame(x\_start = x\_starts, x\_end = x\_ends),

# rows

data\_frame(y\_start = y\_starts, y\_end = y\_ends)) %>%

# So values move left to right, bottom to top

arrange(y\_start, y\_end)

# 400 rows because 20 rows x 20 columns

grid

#> # A tibble: 400 x 4

#> x\_start x\_end y\_start y\_end

#>

#> 1 0 0.05 0 0.05

#> 2 0.05 0.1 0 0.05

#> 3 0.1 0.15 0 0.05

#> 4 0.15 0.2 0 0.05

#> 5 0.2 0.25 0 0.05

#> 6 0.25 0.3 0 0.05

#> 7 0.3 0.35 0 0.05

#> 8 0.35 0.4 0 0.05

#> 9 0.4 0.45 0 0.05

#> 10 0.45 0.5 0 0.05

#> # ... with 390 more rows

We can confirm that the segments in the grid fill out a plot. (I  
randomly color the line segments to make individual ones visible.)

ggplot(grid) +

aes(

x = x\_start, y = y\_start,

xend = x\_end, yend = y\_end,

color = runif(400)) +

geom\_segment(size = 1) +

guides(color = FALSE)

Finally, we need to flip slashes at random. A segment becomes flipped if the  
y\_start and y\_end are swapped. In the code below, we flip the slash in each  
row if a randomly drawn number between 0 and 1 is less than .5. For style, we  
also use theme\_void() to strip away the plotting theme, leaving us with just  
the maze design.

p\_flip <- .5

grid <- grid %>%

arrange(y\_start, y\_end) %>%

mutate(

p\_flip = p\_flip,

flip = runif(length(y\_end)) <= p\_flip,

y\_temp1 = y\_start,

y\_temp2 = y\_end,

y\_start = ifelse(flip, y\_temp2, y\_temp1),

y\_end = ifelse(flip, y\_temp1, y\_temp2)) %>%

select(x\_start:y\_end, p\_flip)

ggplot(grid) +

aes(x = x\_start, y = y\_start, xend = x\_end, yend = y\_end) +

geom\_segment(size = 1, color = "grey20")

last\_plot() + theme\_void()

Now, we wrap all these steps together into a pair of functions.

make\_10\_print\_data <- function(rows = 20, cols = 20, p\_flip = .5) {

x\_points <- seq(0, 1, length.out = cols + 1)

x\_starts <- head(x\_points, -1)

x\_ends <- tail(x\_points, -1)

y\_points <- seq(0, 1, length.out = rows + 1)

y\_starts <- head(y\_points, -1)

y\_ends <- tail(y\_points, -1)

grid <- tidyr::crossing(

data.frame(x\_start = x\_starts, x\_end = x\_ends),

data.frame(y\_start = y\_starts, y\_end = y\_ends))

grid %>%

arrange(y\_start, y\_end) %>%

mutate(

p\_flip = p\_flip,

flip = runif(length(y\_end)) <= p\_flip,

y\_temp1 = y\_start,

y\_temp2 = y\_end,

y\_start = ifelse(flip, y\_temp2, y\_temp1),

y\_end = ifelse(flip, y\_temp1, y\_temp2)) %>%

select(x\_start:y\_end, p\_flip)

}

draw\_10\_print <- function(rows = 20, cols = 20, p\_flip = .5) {

grid <- make\_10\_print\_data(rows = rows, cols = cols, p\_flip = p\_flip)

ggplot(grid) +

aes(x = x\_start, y = y\_start, xend = x\_end, yend = y\_end) +

geom\_segment(size = 1, color = "grey20")

}

**Now the fun part: custom flipping probabilities**

We can vary the probability of flipping the slashes. For example, we can use the  
density of a normal distribution to make flipping more likely for central values  
and less likely for more extreme values.

xs <- seq(0, 1, length.out = 40)

p\_flip <- dnorm(seq(-4, 4, length.out = 40))

ggplot(data.frame(x = xs, y = p\_flip)) +

aes(x, y) +

geom\_line() +

labs(

x = "x position",

y = "p(flipping)",

title = "normal density")

# We repeat p\_flip for each row of the grid

draw\_10\_print(rows = 40, cols = 40, p\_flip = rep(p\_flip, 40)) +

theme\_void()

We can use the cumulative density of the normal distribution so that  
flipping becomes more likely as *x* increases.

xs <- seq(0, 1, length.out = 40)

p\_flip <- pnorm(seq(-4, 4, length.out = 40))

ggplot(data.frame(x = xs, y = p\_flip)) +

aes(x, y) +

geom\_line() +

labs(

x = "x position",

y = "p(flipping)",

title = "cumulative normal")

draw\_10\_print(rows = 40, cols = 40, p\_flip = rep(p\_flip, 40)) +

theme\_void()

The Cauchy distribution is said to have “thicker” tails than the normal  
distribution, so here it shows more flips on the left and right extremes.

xs <- seq(0, 1, length.out = 40)

p\_flip <- dcauchy(seq(-4, 4, length.out = 40))

ggplot(data.frame(x = xs, y = p\_flip)) +

aes(x, y) +

geom\_line() +

labs(

x = "x position",

y = "p(flipping)",

title = "Cauchy density")

draw\_10\_print(rows = 40, cols = 40, p\_flip = rep(p\_flip, 40)) +

theme\_void()

The exponential distribution is a spike that quickly peters out. We can make a  
probability “bowl” by splicing an exponential and a reversed exponential  
together.

# Use flipped exponential densities as probabilities

p\_flip <- c(dexp(seq(0, 4, length.out = 20)),

dexp(seq(4, 0, length.out = 20)))

ggplot(data.frame(x = xs, y = p\_flip)) +

aes(x, y) +

geom\_line() +

labs(

x = "x position",

y = "p(flipping)",

title = "exponential + flipped exponential")

draw\_10\_print(rows = 40, cols = 40, p = rep(p\_flip, 40)) +

theme\_void()

We might have the probabilities increase by 10% from row to row. In the code  
below, I use a simple loop to boost some random probability values by 10% from  
row to row. This gives us nice streaks in the grid as a column starts to flip  
for every row.

boost\_probs <- function(p\_flip, nrows, factor = 1.1) {

output <- p\_flip

for (i in seq\_len(nrows - 1)) {

p\_flip <- p\_flip \* factor

output <- c(output, p\_flip)

}

output

}

draw\_10\_print(cols = 40, rows = 40, p = boost\_probs(runif(40), 40, 1.1)) +

theme\_void()

The probabilities can be anything we like. Here I compute the frequency of  
English alphabet letters as they appear in *Pride and Prejudice* and based the  
flipping probability on those values.

char\_counts <- janeaustenr::prideprejudice %>%

tolower() %>%

stringr::str\_split("") %>%

unlist() %>%

table()

letter\_counts <- char\_counts[letters] %>% as.vector()

p\_letter <- letter\_counts / sum(letter\_counts)

ggplot(data.frame(x = letters, y = p\_letter)) +

aes(x, y, label = x) +

geom\_text() +

labs(

x = NULL,

y = "p(letter)",

title = "letter frequencies in Pride and Prejudice")

draw\_10\_print(cols = 26, rows = 80, p = rep(p\_letter, 80)) +

theme\_void()