Jonasobble

Code ▾

Load Libraries

Hide

```
library(tidyverse)
library(lubridate)
library(primes)
library(httr)
library(jsonlite)
library(magick)

library(packcircles) # to distribute circles (representing images on the canvas)
library(ggforce) # for drawing outer circle
library(ggimage) # for including images in ggplot
```

Introduction

Dobble is based on the freaky fun of finite (https://en.wikipedia.org/wiki/Finite_geometry) projective (https://en.wikipedia.org/wiki/Projective_geometry) geometry.
Exact explanation is a hard TBD.

For now, here's some good resources to understand it (ordered by level of brainfuck):

1. https://puzzlewocky.com/games/the-math-of-spot-it/ (https://puzzlewocky.com/games/the-math-of-spot-it/)
2. http://www.petercollingridge.co.uk/blog/mathematics-toys-and-games/dobble/ (http://www.petercollingridge.co.uk/blog/mathematics-toys-and-games/dobble/)
3. http://www.mathpuzzle.com/MAA/47-Fano/mathgames_05_30_06.html (http://www.mathpuzzle.com/MAA/47-Fano/mathgames_05_30_06.html)

Following is a list of orders for which we can construct Dobble sets, the number of symbols necessary to do so and the number of cards generated (equal to the number of symbols on each card).

Hide

```
projective_plane_orders <- tibble(
    order = sort(c(
      primes::generate_primes(min=1, max=100),
      primes::generate_primes(min=1, max=50)^2,
      primes::generate_primes(min=1, max=20)^3,
      primes::generate_primes(min=1, max=20)^4
    )),
    number_of_cards = order^2 + order + 1,
    number_of_symbols = number_of_cards,
    symbols_per_card = order + 1
  )

projective_plane_orders
```

order	number_of_cards	number_of_symbols	symbols_per_card
<dbl>	<dbl>	<dbl>	<dbl>

order <dbl>	number_of_cards <dbl>	number_of_symbols <dbl>	symbols_per_card <dbl>
2	7	7	3
3	13	13	4
4	21	21	5
5	31	31	6
7	57	57	8
8	73	73	9
9	91	91	10
11	133	133	12
13	183	183	14
16	273	273	17

1-10 of 56 rows Previous **1** 2 3 4 5 6 Next

Init Project

Set Params

Hide

```
# backfill settings
pexels_api_key <- jsonlite::read_json('keys.json')[['pexels_api_key']]

# order of the projective geometry, governing the number of cards generated & images needed
game.order <- 8

page.nrow <- 3
page.ncol <- 2
output_format <- 'png' #'png' or 'pdf'

image_resize.max_side <- 800

backfill_images <- TRUE
```

Hide

```
game.number_of_cards <- projective_plane_orders %>% filter(order == game.order) %>% pull(numb
er_of_cards)
game.number_of_symbols <- projective_plane_orders %>% filter(order == game.order) %>% pull(nu
mber_of_symbols)
game.symbols_per_card <- projective_plane_orders %>% filter(order == game.order) %>% pull(sym
bols_per_card)

cards_per_page <- page.nrow * page.ncol
```

All set for a game of *J(onas)obble* of order 8!
This will generate a set of 73 cards, with 9 symbols each.
(For this we'll need 73 in total, provided into the `/images/input/` folder. If there are less, we'll try to backfill with cat pictures from pexels.com (https://pexels.com).

Prepare images

Set folder structure

Hide

```
image_folder_path <- 'images'
output_folder_path <- 'card_output'

if(!dir.exists(image_folder_path)) { dir.create(image_folder_path) }

image_folder_path.input <- file.path(image_folder_path, 'input')
if(!dir.exists(image_folder_path.input)) { dir.create(image_folder_path.input) }

image_folder_path.backfill <- file.path(image_folder_path, 'backfill')
if(!dir.exists(image_folder_path.backfill)) { dir.create(image_folder_path.backfill) }

image_folder_path.final <- file.path(image_folder_path, 'final')
if(!dir.exists(image_folder_path.final)) { dir.create(image_folder_path.final) }

if(!dir.exists(output_folder_path)) { dir.create(output_folder_path) }
```

Backfill images

If not enough images are provided yet, backfill from Pexels.com (make sure to drop a valid API key into `keys.json`)

Hide

```
# fill the dummy folder
image_files.input <- list.files(path = image_folder_path.input, pattern = '\\.(jpg|jpeg|png)
$', full.names = T, recursive = F)
image_files.backfill <- list.files(path = image_folder_path.backfill, pattern = '\\.(jpg|jpeg
|png)$', full.names = T, recursive = F)

if(backfill_images & !is_empty(pexels_api_key)) {
  backfill.batch_size <- 15

  backfill.max_batches <- ceiling(game.number_of_symbols/backfill.batch_size)
  backfill.num_images_to_fill <- max(game.number_of_symbols - length(image_files.input) - len
gth(image_files.backfill), 0)
  backfill.num_pages_to_query <- ceiling(backfill.num_images_to_fill / backfill.batch_size)

  if(backfill.num_pages_to_query > 0) {
    for(batch_i in tail(1:backfill.max_batches, backfill.num_pages_to_query)) {
      backfill_images_req <- httr::GET(
        paste0('https://api.pexels.com/v1/search?query=cat&per_page=',backfill.batch_size,'&p
age=',batch_i),
        add_headers(Authorization = pexels_api_key)
      )

      backfill_images <- content(backfill_images_req)$photos %>%
        map_dfr(as.data.frame)

      for(i in 1:nrow(backfill_images)) {
        download.file(
          url = ifelse(
            backfill_images$height[i] > 2000 | backfill_images$width[i] > 2000,
            backfill_images$src.large[i],
            backfill_images$src.original[i]
          ),
          destfile = file.path(image_folder_path.backfill, basename(backfill_images$src.origi
nal[i])),
          mode = 'wb',
          #quiet = T
        )
        Sys.sleep(0.2)
      }

    }
  }

  image_files.backfill <- list.files(path = image_folder_path.backfill, pattern = '\\.(jpg|jp
eg|png)$', full.names = T, recursive = F)
}


image_files.src <- tibble(
    image_file_path = c(image_files.input, image_files.backfill),
    image_type = c(rep('input', length(image_files.input)), rep('backfill', length(image_file
s.backfill)))
  ) %>%
  bind_cols(
    magick::image_read(.$image_file_path) %>% magick::image_info()
  )
```

```
image_files.final <- image_files.src %>%
  select (
    image_file_path.src = image_file_path, image_type
  )

for(i in 1:nrow(image_files.final)) {
  tryCatch({
    img <- magick::image_read(image_files.final$image_file_path.src[i])
    img.resized <- image_resize(img, paste(image_resize.max_side,image_resize.max_side,sep=
'x'))

    image_file_path.resized <- file.path(image_folder_path.final, basename(image_files.final
$image_file_path.src[i]))

    magick::image_write(img.resized, image_file_path.resized)
    # print(sprintf('resized %s to the dimensions %sx%s px', basename(image_files.final$image
_file_path.src[i]), magick::image_info(img.resized)$width, magick::image_info(img.resized)$he
ight))

    image_files.final$image_file_path.resized[i] <- image_file_path.resized
  })
}

image_files.final <- image_files.final %>%
  filter(!is.na(image_file_path.resized)) %>%
  bind_cols(
    magick::image_read(.$image_file_path.resized) %>% magick::image_info()
  )

print(sprintf('resized %s images from the input & backfill folders to max dimensions %sx%s p
x', nrow(image_files.final),image_resize.max_side,image_resize.max_side))
```

```
[1] "resized 73 images from the input & backfill folders to max dimensions 800x800 px"
```

We've now gathered & resized 73 image files to work into the game

Generator functions

Dobble Permutations

This function will return a list of symbol permutations (representing individual game-cards) for a given `order` .

Hide

```
# taken code from https://math.stackexchange.com/a/1956107
generate_dobble_permutations <- function(order) {
  card_symbols <- list()

  # build the initial set of cards as per
  for(i in 0:order) {
    symbol_indices <- c(1, 1:order + i*order + 1)

    card_symbols[[paste(1,i+1,sep='_')]] <- symbol_indices
  }

  # build the rest of the deck
  for(i in seq(0,order-1)) {
    for(j in seq(0,order-1)) {
      k <- seq(0,order-1)
      symbol_indices <- c(i+1+1, order+1 + order*k +(i*k+j)%%order +1)
      card_symbols[[paste(i+2,j+1,sep='_')]] <- symbol_indices
    }
  }

  return(card_symbols)
}
```

Distribute Images

This function generates coordinates on & within a `radius = 1` circle where we can position our images. Based on the patterns of sunflower seeds (https://momath.org/home/fibonacci-numbers-of-sunflower-seed-spirals/).

Hide

```
# https://stackoverflow.com/a/28572551/1335174
generate_image_coordinates.sunflower <- function(n, alpha = 2, geometry = c('planar','geodesi
c')) {
  radius <- function(k,n,b) {
    ifelse(k > n-b, 1, sqrt(k-1/2)/sqrt(n-(b+1)/2))
  }

  b <- round(alpha*sqrt(n))  # number of boundary points
  phi <- (sqrt(5)+1)/2  # golden ratio

  r <- radius(1:n,n,b)
  #theta <- 2*pi*1:n/phi^2
  theta <- 1:n * ifelse(geometry[1] == 'geodesic', 360*phi, 2*pi/phi^2)

  tibble(
    symbol_on_card = seq(n),
    x = r*cos(theta),
    y = r*sin(theta)
  )
}

generate_image_coordinates.circlepacking <- function(image_file_paths, margin = 0) {
  image_data <- magick::image_read(image_file_paths) %>%
    magick::image_info() %>%
    mutate(
      image_file_path = image_file_paths,
      diagonal = sqrt(height^2 + width^2),
      radius = diagonal/2 + margin, # add a margin to the radius
    ) %>%
    arrange(desc(radius))

  image_data %>%
    select(image_file_path, height, width, diagonal, radius) %>%
    bind_cols(
      packcircles::circleProgressiveLayout(.$radius, sizetype = 'radius')
    ) %>%
    mutate(
      radius = diagonal/2
    )
}
```

Create Dobble cards

Calculate Symbol Permutations

Hide

```
dobble_cards.permutations <- generate_dobble_permutations(game.order)
```

We now have a list of `game.number_of_cards` cards with generic numbers representing the 73 symbols we will use.
Let’s look at the top few:

Hide

```
head(dobble_cards.permutations)
```

```
$`1_1`
[1] 1 2 3 4 5 6 7 8 9

$`1_2`
[1]  1 10 11 12 13 14 15 16 17

$`1_3`
[1]  1 18 19 20 21 22 23 24 25

$`1_4`
[1]  1 26 27 28 29 30 31 32 33

$`1_5`
[1]  1 34 35 36 37 38 39 40 41

$`1_6`
[1]  1 42 43 44 45 46 47 48 49
```

We'll break this up into a dataframe where we will have one row per symbol on each card.
Also, we're assigning the prepared images by using the symbol-number as an index.

Hide

```
dobble_cards.df <- dobble_cards.permutations %>%
  map_dfr(.f = function(indices){
    tibble(
      symbol = indices,
      symbol_on_card = seq(symbol)
    )},
    .id = 'card'
  ) %>%
  mutate(
    card_generation_block = str_extract(card, '^\\d+(?=_)'),
    card_id = as.integer(as_factor(card)),
    image_file_path = image_files.final$image_file_path.resized[symbol],
    image_name = basename(image_file_path)
  )

dobble_cards.df
```

card <chr>	symbol <dbl>	symbol_on_card <int>	card_generation_block <chr>	card_id <int>
1_1	1	1	1	1
1_1	2	2	1	1
1_1	3	3	1	1
1_1	4	4	1	1
1_1	5	5	1	1
1_1	6	6	1	1

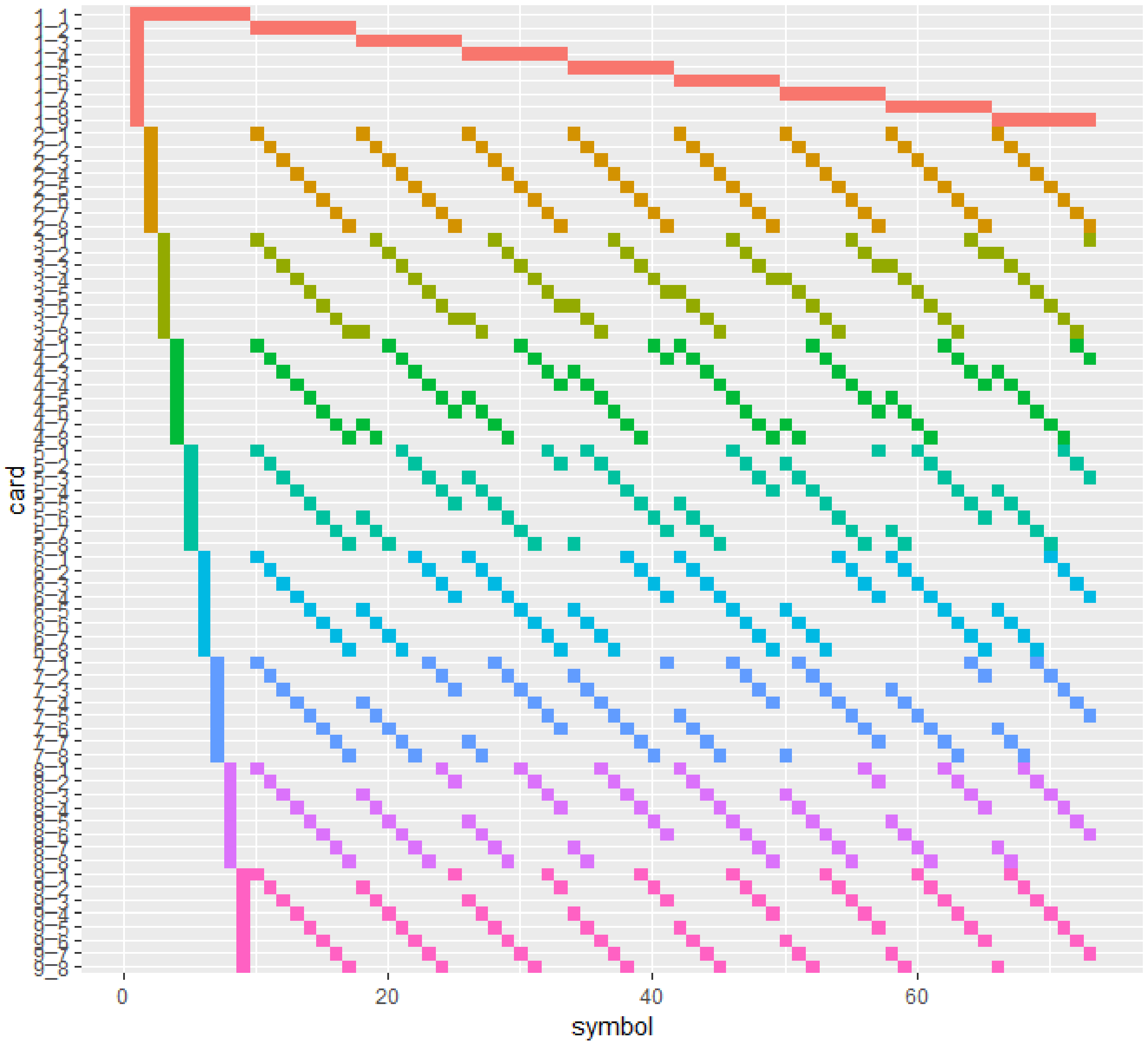
card <chr>	symbol <dbl>	symbol_on_card <int>	card_generation_block <chr>	card_id <int>
1_1	7	7	1	1
1_1	8	8	1	1
1_1	9	9	1	1
1_2	1	1	1	2

1-10 of 657 rows | 1-5 of 7 columns Previous **1** 2 3 4 5 6 ... 10 Next

To visualize the distribution of images onto the cards, we can plot a matrix of cards (y-axis) and the symbols printed on them (x-axis).
We can see that there is always one symbol that is shared among any two cards, but not more.

Hide

```
dobble_cards.df %>%
  ggplot() +
  geom_raster(
    aes(
      x = symbol,
      y = fct_rev(as_factor(card)),
      fill = card_generation_block
    )
  ) +
  scale_color_discrete(aesthetics = c('fill','color'), guide = F) +
  coord_fixed() +
  labs(
    x = 'symbol',
    y = 'card'
  )
```



Distribute Images on Cards

Finally, we need to figure out where to place the image files on the generated cards (which will be plotted as charts using ggplot).
We will use a circlepacking approach, where we identify for each image a circle into which it can fit and then densely pack these around the (0,0) center of the plot.

Hide

```
image_margin <- image_resize.max_side * 0.025 # for 1000px resized images, this is 5px each,
 so 10px spacing in between

# calculate the image coordinates for each card
dobble_cards.image_coords <- dobble_cards.df %>%
  group_by(card_id) %>%
  group_map(~generate_image_coordinates.circlepacking(.$image_file_path, margin = image_margi
n)) %>%
  # unfortunately, the packcircles methods don't center the results around (0,0), so we corre
ct their x & y values here
  mutate(
    tmp.real_circle_center.x = (min(x-radius) + max(x+radius))/2,
    tmp.real_circle_center.y = (min(y-radius) + max(y+radius))/2,

    x = x - tmp.real_circle_center.x,
    y = y - tmp.real_circle_center.y
  ) %>%
  ungroup() %>%
  select(-starts_with('tmp.'))
```

We've laid out all of the images on our cards, given them a margin of 20 px each, so that they would have a little space in between them.
Having prepared these "circles", we can check across all cards for the most distant circle that still needs to be on a card.
That will set the outer radius which we will draw to be able to cut out equal sized round playing cards.

Hide

```
dobble_cards.card_radius <- dobble_cards.image_coords %>%
  summarise(
    outer_radius = max(sqrt(x^2+y^2) + radius)
  ) %>%
  pull(outer_radius) %>%
  ceiling() %>%
  {. + 2}
```

To accommodate all pictures safely, the cards will all have a radius of 2252 px.

We can plot this without the actual images, to see a bit more clearly how this will look like for the first 4 cards.

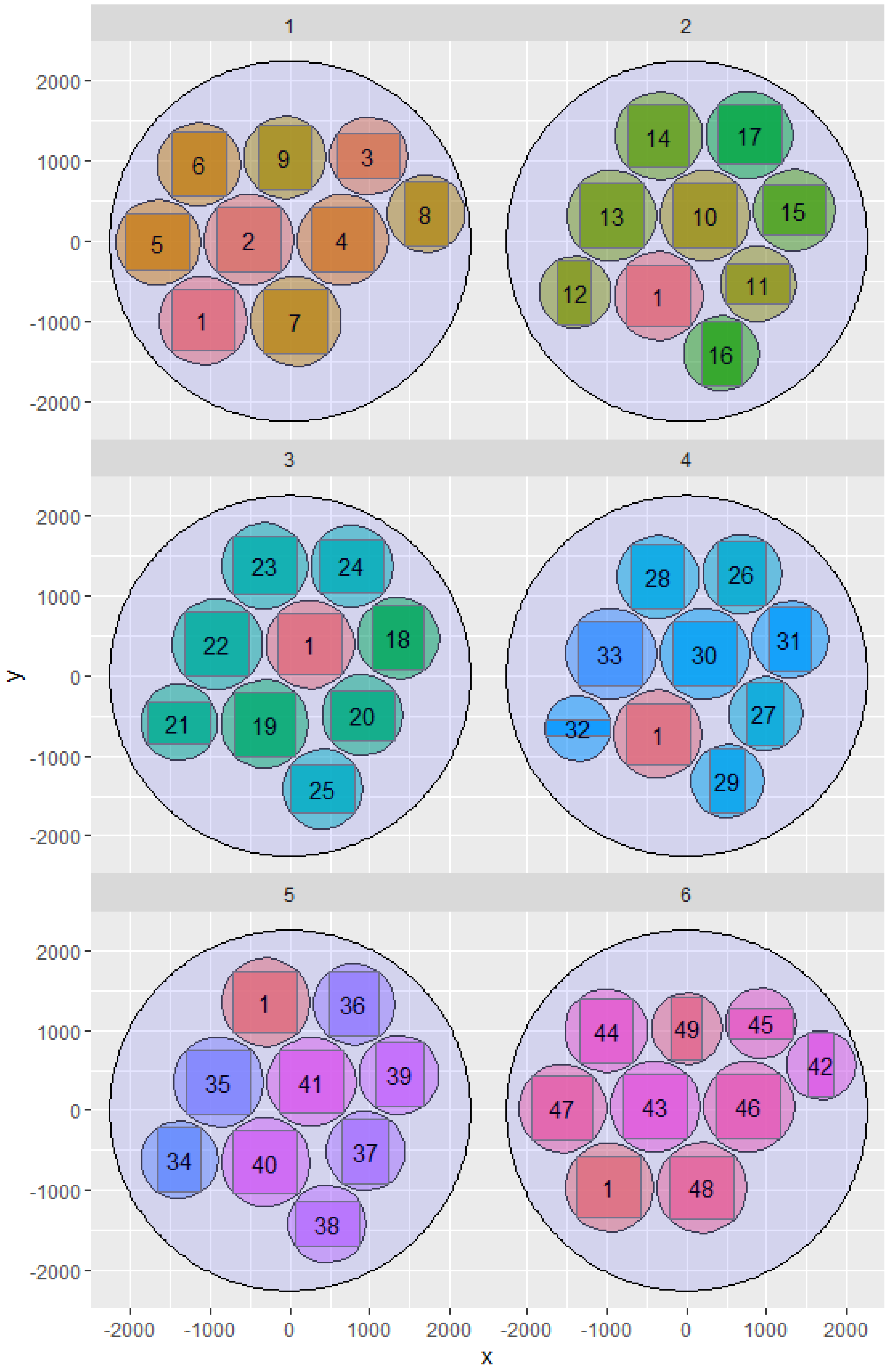
Hide

```
dobble_cards.image_boundaries.circles <- dobble_cards.image_coords %>%
  mutate(
    id = paste(card_id, image_file_path, sep = '_')
  ) %>%
  select(x, y, radius, id) %>%
  packcircles::circleLayoutVertices(idcol = 'id') %>%
  mutate(
    card_id = as.integer(str_extract(id, '^\\d+(?=_)')),
    image_file_path = str_extract(id, '(?<=^\\d{1,5}_).+$')
  )

dobble_cards.image_boundaries.squares <- dobble_cards.image_coords %>%
  mutate(
    x0 = x - 0.5*width,
    x1 = x + 0.5*width,
    y0 = y - 0.5*height,
    y1 = y + 0.5*height
  )
```

Hide

```r
ggplot() +
  geom_polygon(
    data = dobble_cards.image_boundaries.circles %>%
      filter(card_id <= cards_per_page),
    aes(
      x, y,
      group = id,
      fill = image_file_path
    ),
    color = 'grey25',
    alpha = 0.5
  ) +
  geom_rect(
    data = dobble_cards.image_boundaries.squares %>%
      filter(card_id <= cards_per_page),
    aes(
      xmin = x0, xmax = x1,
      ymin = y0, ymax = y1,
      fill = image_file_path
    ),
    color = 'grey50',
    alpha = 0.8
  ) +
  geom_text(
    data = dobble_cards.image_coords %>%
      filter(card_id <= cards_per_page) %>%
      left_join(dobble_cards.df, by = c('card_id', 'image_file_path')),
    aes(
      x, y,
      label = symbol
    )
  ) +
  ggforce::geom_circle(
    data = tibble(card_id = 1:cards_per_page),
    aes(
      x0 = 0,
      y0 = 0,
      r = dobble_cards.card_radius
    ),
    fill = 'blue',
    alpha = 0.1
  ) +
  scale_color_discrete(aesthetics = c('color','fill'), guide = F) +
  coord_fixed() +
  facet_wrap(~card_id, nrow = page.nrow, ncol = page.ncol) +
  theme(panel.spacing=unit(0, 'mm'))
```



Finalize Card Set

Now we'll bind the calculated image coordinates to our previously generated permutation data and receive a data table that tells us exactly which image will go on which card *and where*.

Hide

```
dobble_cards.final <- dobble_cards.df %>%
  left_join(
    dobble_cards.image_coords,
    by = c('card_id', 'image_file_path')
  )
```

Using that, we'll generate 13 pages in png format (holding 6 cards each) into the output folder card_output.

Hide

```
pages <- tibble(
    page_i = 1:ceiling(game.number_of_cards/cards_per_page),
    cards.first_id = (page_i-1)*cards_per_page +1,
    cards.last_id = pmin(page_i*cards_per_page, game.number_of_cards)
  )


for(page_i in 1:nrow(pages)) {
  page <- pages[page_i,]

  page_filename = paste(strftime(lubridate::now(), '%Y-%m-%d'), sprintf('jonasobble_page_%02
d',page_i), sep = '_')
  page_title = sprintf('J(onas)obble Game, page %s of %s', page_i, nrow(pages))

  page_plot <- ggplot() +
    ggforce::geom_circle(
      data = tibble(card_id = seq(page$cards.first_id, page$cards.first_id+cards_per_page-1
)),
      aes(
        x0 = 0,
        y0 = 0,
        r = dobble_cards.card_radius
      ),
      #fill = 'blue',
      #alpha = 0.2
    ) +
    ggimage::geom_image(
      data = dobble_cards.final %>%
        filter(
          card_id >= page$cards.first_id,
          card_id <= page$cards.last_id
        ),
      aes(
        x = x,
        y = y,
        image = image_file_path,
        size=I(0.5*width/dobble_cards.card_radius)
      )
    ) +
    coord_fixed() +
    facet_wrap(~card_id, nrow = page.nrow, ncol = page.ncol, strip.position = 'left') +
    theme_void()

  if(output_format == 'pdf') {
    pdf(
      file = file.path(output_folder_path, paste0(page_filename, '.pdf')),
      title = pdf_title,
      paper = 'a4',
      width = 8, height = 11.5,
      bg = 'transparent',
      compress = F
    )
    print(page_plot)
    dev.off()

  } else if(output_format == 'png') {
    ggsave(
      filename = paste0(page_filename,'.png'),
      device = 'png',
      width = 210, height = 297, units = 'mm', # a4
      dpi = 320,
      path = output_folder_path
    )
  }

  print(paste('printed page',page_i))
}
```

```
[1] "printed page 1"
[1] "printed page 2"
[1] "printed page 3"
[1] "printed page 4"
[1] "printed page 5"
[1] "printed page 6"
[1] "printed page 7"
[1] "printed page 8"
[1] "printed page 9"
[1] "printed page 10"
[1] "printed page 11"
[1] "printed page 12"
[1] "printed page 13"
```