Package 'aRtsy'

May 16, 2025

Title Generative Art with 'ggplot2'

Description Provides algorithms for creating artworks in the 'ggplot2' language that incorporate some form of randomness.

Version 1.0.1

Date 2025-05-16

BugReports https://github.com/koenderks/aRtsy/issues

URL https://koenderks.github.io/aRtsy/,
https://github.com/koenderks/aRtsy,
https://twitter.com/aRtsy_package,
https://mastodon.social/@aRtsy_package

Suggests testthat (>= 3.0.0)

Imports ambient, e1071, ggplot2 (>= 3.4.0), FNN, randomForest, Rcpp, scales, stats

LinkingTo Rcpp, RcppArmadillo

Language en-US

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

NeedsCompilation yes

Author Koen Derks [aut, cre]

Maintainer Koen Derks <koen-derks@hotmail.com>

Repository CRAN

Date/Publication 2025-05-16 08:50:28 UTC

Contents

aRtsy-package *aRtsy — Generative Art using* ggplot2

Description

aRtsy aims to make generative art accessible to the general public in a straightforward and standardized manner. The package provides algorithms for creating artworks that incorporate some form of randomness and are dependent on the set seed. Each algorithm is implemented in a separate function with its own set of parameters that can be tweaked.

For documentation on aRtsy itself, including the manual and user guide for the package, worked examples, and other tutorial information visit the package website.

Author(s)

Koen Derks (maintainer, author) <koen-derks@hotmail.com>

Please use the citation provided by R when citing this package. A BibTex entry is available from `citation("aRtsy")`.

See Also

Useful links:

- The twitter feed to check the artwork of the day.
- The issue page to submit a bug report or feature request.

canvas_ant *Draw Langton's Ant*

Description

This function draws Langton's Ant on a canvas. Langton's Ant is a two-dimensional cellular automaton that is named after its creator, Chris Langton. See the Details section for more specific information about the algorithm used in this function.

Usage

```
canvas_ant(
  colors,
  background = "#fafafa",
  iterations = 1000000,
  resolution = 500
)
```

Arguments

colors	a character (vector) specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
iterations	a positive integer specifying the number of iterations of the algorithm.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Details

The algorithm for Langton's Ant involves the following steps:

- Set up a two-dimensional grid of cells, where each cell can either be "colored" or "non-colored." The initial state of the grid is usually a single non-colored cell in the center of the grid.
- Place an "ant" on the grid at the position of the initial non-colored cell. The ant can move in four directions: up, down, left, or right.
- At each step of the algorithm, the ant examines the color of the cell it is currently on. If the cell is non-colored, the ant turns 90 degrees clockwise, colors the cell, and moves forward one unit.
- If the cell is colored, the ant turns 90 degrees counterclockwise, uncolors the cell, and moves forward one unit.
- The ant continues to move around the grid, following these rules at each step. If a certain number of iterations has passed, the ant chooses a different color which corresponds to a different combination of these rules.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Langtons_ant

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_ant(colors = colorPalette("house"))
```

`canvas_blacklight` *Draw Blacklights*

Description

This function draws Blacklights on a canvas using a Support Vector Machine (SVM) algorithm. SVM's are a type of supervised learning algorithm that can be used for classification and regression purposes. The main goal of the SVM technique is to find a hyperplane (decision boundary) that best separates the values in the training dataset. This function draws the predictions from the SVM algorithm fitted on a randomly generated continuous training data set.

Usage

```
canvas_blacklight(
  colors,
  n = 1000,
  resolution = 500
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`n`	a positive integer specifying the number of random data points to generate.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Support-vector_machine

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_blacklight(colors = colorPalette("tuscany2"))
```

`canvas_chladni` *Draw Chladni Figures*

Description

This function draws Chladni figures on a canvas. Named after Ernst Chladni, an 18th century physicist who first discovered them, Chladni figures are patterns that arise from the vibrations of a two-dimensional plate, typically covered with a thin layer of sand or powder. The Chladni figures are created by varying the frequency of vibration applied to the plate. In this implementation, the grid underneath the plate can be transformed using a domain warping technique. The basic idea behind domain warping is to apply a series of transformations to the input grid to create a more complex and interesting output.

Usage

```
canvas_chladni(
  colors,
  waves = 5,
  warp = 0,
  resolution = 500,
  angles = NULL,
  distances = NULL,
  flatten = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`waves`	a character specifying the number of randomly sampled waves, or an integer vector of waves to be summed.
`warp`	a numeric value specifying the maximum warping distance for each point. If `warp = 0` (the default), no warping is performed.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

angles	optional, a resolution x resolution matrix containing the angles for the warp, or a character indicating the type of noise to use (svm, knn, rf, perlin, cubic, simplex, or worley). If NULL (the default), the noise type is chosen randomly.
distances	optional, a resolution x resolution matrix containing the distances for the warp, or a character indicating the type of noise to use (svm, knn, rf, perlin, cubic, simplex, or worley). If NULL (the default), the noise type is chosen randomly.
flatten	logical, should colors be flattened after being assigned to a point.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(2)

# Simple example
canvas_chladni(colors = colorPalette("origami"))

# Advanced example
canvas_chladni(
  colors = colorPalette("lava"),
  waves = c(1, 2, 3, 9),
  warp = 1
)
```

---

canvas_circlemap *Draw a Circle Map*

Description

This function draws a circle map on a canvas. A circle map is a nonlinear dynamic system that can exhibit a phenomenon known as Arnold's tongue: a visualization of the frequency-locking behavior of a nonlinear oscillator with a periodic external force. The tongue is a region in the parameter space of the oscillator where the frequency of the oscillator matches the frequency of the external force. The tongue appears as a series of tongues of varying widths and shapes that can extend into regions of the parameter space where the frequency locking does not occur.

Usage

```
canvas_circlemap(
  colors,
  left = 0,
  right = 12.56,
  bottom = 0,
  top = 1,
  iterations = 10,
  resolution = 1500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
left	a value specifying the minimum location on the x-axis.
right	a value specifying the maximum location on the x-axis.
bottom	a value specifying the minimum location on the y-axis.
top	a value specifying the maximum location on the y-axis.
iterations	a positive integer specifying the number of iterations of the algorithm.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Arnold_tongue

https://linas.org/art-gallery/circle-map/circle-map.html

See Also

colorPalette

Examples

```
canvas_circlemap(colors = colorPalette("dark2"))
```

canvas_cobweb *Draw Cobwebs*

Description

This function draws a cobweb on the canvas. The cobweb consists of many Fibonacci spirals shifted by random noise from a normal distribution. A Fibonacci spiral is a logarithmic spiral that is derived from the Fibonacci sequence, a mathematical sequence where each number is the sum of the two preceding ones. The spiral is created by connecting the corners of squares that are sized according to the Fibonacci sequence. Specifically, if we draw a sequence of squares with side lengths of 1, 1, 2, 3, 5, 8, 13, and so on, each square can be arranged so that it is tangent to the previous square at a corner. When we connect these corners with a smooth curve, the resulting shape is the Fibonacci spiral.

Usage

```
canvas_cobweb(
  colors,
  background = "#fafafa",
  lines = 300,
  iterations = 100
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
lines	the number of lines to draw.
iterations	the number of iterations of the algorithm.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_cobweb(colors = colorPalette("neon1"), background = "black")
```

`canvas_collatz` *Draw Collatz Sequences*

Description

This function draws the Collatz conjecture on a canvas. The conjecture of the Collatz sequence is that no matter what positive integer is chosen as the starting point of the sequence, the sequence will eventually reach the number 1. This conjecture has been verified for all starting integers up to very large numbers, but it has not been proven mathematically. Despite its simple rule, the sequence can produce long and complicated chains of numbers before eventually reaching 1. See the `Details` section for more specific information about the algorithm used in this function.

Usage

```
canvas_collatz(
  colors,
  background = "#fafafa",
  n = 200,
  angle.even = 0.0075,
  angle.odd = 0.0145,
  side = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the background.
`n`	a positive integer specifying the number of random starting integers to use for the lines. Can also be a vector of numbers to use as starting numbers.
`angle.even`	a value specifying the angle (in radials) to use in bending the sequence at each even number.
`angle.odd`	a value specifying the angle (in radials) to use in bending the sequence at each odd number.
`side`	logical. Whether to put the artwork on its side.

Details

The Collatz sequence, also known as the 3n+1 problem, is a sequence of numbers generated by the following rule:

- Start with any positive integer n.
- If n is even, divide it by 2.
- If n is odd, multiply it by 3 and add 1.
- Repeat this process with the new value of n, generating a new number in the sequence.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://nl.wikipedia.org/wiki/Collatz_Conjecture

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_collatz(colors = colorPalette("tuscany3"))
```

---

canvas_diamonds *Draw Diamonds*

Description

This function draws diamonds on a canvas and (optionally) places two lines behind them. The diamonds can be transparent or have a random color sampled from the input.

Usage

```
canvas_diamonds(
  colors,
  background = "#fafafa",
  col.line = "black",
  radius = 10,
  alpha = 1,
  p = 0.2,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
col.line	a character specifying the color of the diamond borders.
radius	a positive value specifying the radius of the diamonds.
alpha	a value specifying the transparency of the diamonds. If NULL (the default), added layers become increasingly more transparent.
p	a value specifying the probability of drawing an empty diamond.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_diamonds(colors = colorPalette("tuscany1"))
```

canvas_flame *Draw a Fractal Flame*

Description

This function implements the fractal flame algorithm.

Usage

```
canvas_flame(
  colors,
  background = "#000000",
  iterations = 1000000,
  variations = 0,
  symmetry = 0,
  blend = TRUE,
  weighted = FALSE,
  post = FALSE,
  final = FALSE,
  extra = FALSE,
  display = c("colored", "logdensity"),
  zoom = 1,
  resolution = 1000,
  gamma = 1
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
iterations	a positive integer specifying the number of iterations of the algorithm. Using more iterations results in images of higher quality but also increases the computation time.
variations	an integer (vector) with a minimum of 0 and a maximum of 48 specifying the variations to be included in the flame. The default `0` includes only a linear variation. Including multiple variations (e.g., `c(1, 2, 3)`) increases the computation time. See the details section for more information about possible variations.
symmetry	an integer with a minimum of -6 and a maximum of 6 indicating the type of symmetry to include in the flame. The default `0` includes no symmetry. Including symmetry decreases the computation time as a function of the absolute `symmetry` value. See the details section for more information about possible symmetries.
blend	logical. Whether to blend the variations (`TRUE`) or pick a unique variation in each iteration (`FALSE`). `blend = TRUE` increases computation time as a function of the number of included variations.

weighted	logical. Whether to weigh the functions and the variations (TRUE) or pick a function at random and equally weigh all variations (FALSE). weighted = TRUE significantly increases the computation time.
post	logical. Whether to apply a post transformation in each iteration.
final	logical. Whether to apply a final transformation in each iteration.
extra	logical. Whether to apply an additional post transformation after the final transformation. Only has an effect when final = TRUE.
display	a character indicating how to display the flame. colored (the default) displays colors according to which function they originate from. logdensity plots a gradient using the log density of the pixel count.
zoom	a positive value specifying the amount of zooming.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution does not increases the computation time of this algorithm.
gamma	a numeric value specifying the gamma correction (only used when display = "colored"). Larger values result in brighter images and vice versa.

Details

The variation argument can be used to include specific variations into the flame. See the appendix in the references for examples of all variations. Possible variations are:

- 0: Linear (default)
- 1: Sinusoidal
- 2: Spherical
- 3: Swirl
- 4: Horsehoe
- 5: Polar
- 6: Handkerchief
- 7: Heart
- 8: Disc
- 9: Spiral
- 10: Hyperbolic
- 11: Diamond
- 12: Ex
- 13: Julia
- 14: Bent
- 15: Waves
- 16: Fisheye
- 17: Popcorn
- 18: Exponential
- 19: Power

- 20: Cosine
- 21: Rings
- 22: Fan
- 23: Blob
- 24: PDJ
- 25: Fan2
- 26: Rings2
- 27: Eyefish
- 28: Bubble
- 29: Cylinder
- 30: Perspective
- 31: Noise
- 32: JuliaN
- 33: JuliaScope
- 34: Blur
- 35: Gaussian
- 36: RadialBlur
- 37: Pie
- 38: Ngon
- 39: Curl
- 40: Rectangles
- 41: Arch
- 42: Tangent
- 43: Square
- 44: Rays
- 45: Blade
- 46: Secant
- 47: Twintrian
- 48: Cross

The `symmetry` argument can be used to include symmetry into the flame. Possible options are:

- 0: No symmetry (default)
- -1: Dihedral symmetry
- 1: Two-way rotational symmetry
- (-)2: (Dihedral) Three-way rotational symmetry
- (-)3: (Dihedral) Four-way rotational symmetry
- (-)4: (Dihedral) Five-way rotational symmetry
- (-)5: (Dihedral) Six-way rotational symmetry
- (-)6: (Dihedral) Snowflake symmetry

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://flam3.com/flame_draves.pdf

See Also

colorPalette

Examples

```
set.seed(3)

# Simple example, linear variation, relatively few iterations
canvas_flame(colors = c("dodgerblue", "green"), variations = 0)

# Simple example, linear variation, dihedral symmetry
canvas_flame(colors = c("hotpink", "yellow"), variations = 0, symmetry = -1, iterations = 1e7)

# Advanced example (no-blend, weighted, sinusoidal and spherical variations)
canvas_flame(
  colors = colorPalette("origami"), variations = c(1, 2),
  blend = FALSE, weighted = TRUE, iterations = 1e8
)

# More iterations give much better images
set.seed(123)
canvas_flame(colors = c("red", "blue"), iterations = 1e8, variations = c(10, 17))
```

---

canvas_flow *Draw A Flow Field*

Description

This function draws flow fields on a canvas. The algorithm simulates the flow of points through a field of angles which can be set manually or generated from the predictions of a supervised learning method (i.e., knn, svm, random forest) trained on randomly generated data.

Usage

```
canvas_flow(
  colors,
  background = "#fafafa",
  lines = 500,
  lwd = 0.05,
  iterations = 100,
  stepmax = 0.01,
  outline = c("none", "circle", "square"),
  polar = FALSE,
  angles = NULL
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
lines	the number of lines to draw.
lwd	expansion factor for the line width.
iterations	the maximum number of iterations for each line.
stepmax	the maximum proportion of the canvas covered in each iteration.
outline	character. Which outline to use for the artwork. Possible options are none (default), circle or square.
polar	logical. Whether to draw the flow field with polar coordinates.
angles	optional, a 200 x 200 matrix containing the angles in the flow field, or a character indicating the type of noise to use (svm, knn, rf, perlin, cubic, simplex, or worley). If NULL (the default), the noise type is chosen randomly.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://tylerxhobbs.com/essays/2020/flow-fields

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_flow(colors = colorPalette("dark2"))

# Outline example
canvas_flow(
  colors = colorPalette("vrolik1"), lines = 10000,
  outline = "circle", iterations = 10, angles = "svm"
)

# Polar example
canvas_flow(
  colors = colorPalette("vrolik2"), lines = 300,
  lwd = 0.5, polar = TRUE
)

# Advanced example
angles <- matrix(0, 200, 200)
angles[1:100, ] <- seq(from = 0, to = 2 * pi, length = 100)
angles[101:200, ] <- seq(from = 2 * pi, to = 0, length = 100)
angles <- angles + rnorm(200 * 200, sd = 0.1)
canvas_flow(
  colors = colorPalette("tuscany1"), background = "black",
  angles = angles, lwd = 0.4, lines = 1000, stepmax = 0.001
)
```

---

canvas_forest *Draw a Random Forest*

Description

This function draws the predictions from a random forest algorithm trained on randomly generated categorical data.

Usage

```
canvas_forest(
  colors,
  n = 1000,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
n	a positive integer specifying the number of random data points to generate.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Random_forest

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_forest(colors = colorPalette("jungle"))
```

---

canvas_function *Draw Functions*

Description

This function paints functions with random parameters on a canvas.

Usage

```
canvas_function(
  colors,
  background = "#fafafa",
  by = 0.01,
  polar = TRUE,
  formula = NULL
)
```

Arguments

colors	a string specifying the color used for the artwork.
background	a character specifying the color used for the background.
by	a value specifying the step size between consecutive points.
polar	logical. Whether to draw the function with polar coordinates.
formula	optional, a named list with 'x' and 'y' as structured in the example. If NULL (default), chooses a function with random parameters.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://github.com/cutterkom/generativeart

See Also

colorPalette

Examples

```
set.seed(10)

# Simple example
canvas_function(colors = colorPalette("tuscany1"))

# Advanced example
formula <- list(
  x = quote(x_i^2 - sin(y_i^2)),
  y = quote(y_i^3 - cos(x_i^2))
)
canvas_function(colors = "firebrick", formula = formula)
```

canvas_gemstone *Draw Gemstones*

Description

This function draws the predictions from a k-nearest neighbors algorithm trained on randomly generated continuous data.

Usage

```
canvas_gemstone(
  colors,
  n = 1000,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
n	a positive integer specifying the number of random data points to generate.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_gemstone(colors = colorPalette("dark3"))
```

`canvas_lissajous` *Draw a Lissajous Curve*

Description

This function draws lissajous curves with points connected via a k-nearest neighbor approach.

Usage

```
canvas_lissajous(
  colors,
  background = "#000000",
  iterations = 2,
  neighbors = 50,
  noise = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the background.
`iterations`	a positive integer specifying the number of iterations of the algorithm.
`neighbors`	a positive integer specifying the number of neighbors a block considers when drawing the connections.
`noise`	logical. Whether to add perlin noise to the coordinates of the nodes.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Lissajous_curve

See Also

`colorPalette`

Examples

```
set.seed(13)

# Simple example
canvas_lissajous(colors = colorPalette("blossom"))
```

`canvas_mandelbrot` *Draw the Mandelbrot Set*

Description

This function draws the Mandelbrot set and other related fractal sets on the canvas.

Usage

```
canvas_mandelbrot(
  colors,
  iterations = 100,
  zoom = 1,
  set = c("mandelbrot", "multibrot", "julia", "ship"),
  left = -2.16,
  right = 1.16,
  bottom = -1.66,
  top = 1.66,
  resolution = 500
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`iterations`	a positive integer specifying the number of iterations of the algorithm.
`zoom`	a positive value specifying the amount of zoom to apply.
`set`	a character indicating which fractal set to draw. Possible options are `mandelbrot` for the Mandelbrot set, `multibrot` for variations of the Mandelbrot set (aka the Multibrot sets), `julia` for the Julia set and `ship` for the Burning ship set.
`left`	a value specifying the minimum location on the x-axis.
`right`	a value specifying the maximum location on the x-axis.
`bottom`	a value specifying the minimum location on the y-axis.
`top`	a value specifying the maximum location on the y-axis.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Mandelbrot_set

See Also

colorPalette

Examples

```
canvas_mandelbrot(colors = colorPalette("tuscany1"), set = "mandelbrot")
canvas_mandelbrot(colors = colorPalette("flag"), set = "julia", zoom = 2)
```

---

canvas_maze *Draw Mazes*

Description

This function draws a maze on a canvas.

Usage

```
canvas_maze(
  color = "#fafafa",
  walls = "black",
  background = "#fafafa",
  resolution = 20,
  polar = FALSE
)
```

Arguments

color	a character specifying the color used for the artwork.
walls	a character specifying the color used for the walls of the maze.
background	a character specifying the color used for the background.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.
polar	logical, whether to use polar coordinates. Warning, this increases display and saving time dramatically.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://github.com/matfmc/mazegenerator

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_maze(color = "#fafafa")
```

---

canvas_mesh *Draw Meshes*

Description

This function draws one or more rotating circular morphing meshes on the canvas.

Usage

```
canvas_mesh(
  colors,
  background = "#fafafa",
  transform = c("perlin", "fbm", "simplex", "cubic",
                "worley", "knn", "rf", "svm"),
  lines = 500,
  iterations = 500,
  mixprob = 0
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background (and the hole).
transform	a character specifying the type of transformation to use for the radius.
lines	an integer specifying the number of lines to darw.
iterations	a positive integer specifying the number of iterations of the algorithm.
mixprob	a value between 0 and 1 specifying the probability of a line segment getting another color.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://web.archive.org/web/20231130151102/http://rectangleworld.com/blog/archives/462

See Also

colorPalette

Examples

```
set.seed(2)

# Simple example
canvas_mesh(colors = colorPalette("origami"))
```

---

canvas_mosaic *Draw Moisaics*

Description

This function draws the predictions from a k-nearest neighbors algorithm trained on randomly generated categorical data.

Usage

```
canvas_mosaic(
  colors,
  n = 1000,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
n	a positive integer specifying the number of random data points to generate.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_mosaic(colors = colorPalette("retro2"))
```

canvas_nebula *Draw Nebulas*

Description

This function creates an artwork from randomly generated k-nearest neighbors noise.

Usage

```
canvas_nebula(
  colors,
  k = 50,
  n = 500,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
k	a positive integer specifying the number of nearest neighbors to consider.
n	a positive integer specifying the number of random data points to generate.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_nebula(colors = colorPalette("tuscany1"))
```

canvas_petri *Draw Petri Dish Colonies*

Description

This function uses a space colony algorithm to draw Petri dish colonies.

Usage

```
canvas_petri(
  colors,
  background = "#fafafa",
  dish = "black",
  attractors = 1000,
  iterations = 15,
  hole = 0
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background (and the hole).
dish	a character specifying the color used for the Petri dish.
attractors	an integer specifying the number of attractors.
iterations	a positive integer specifying the number of iterations of the algorithm.
hole	a value between 0 and 0.9 specifying the hole size in proportion to the dish.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://medium.com/@jason.webb/space-colonization-algorithm-in-javascript-6f683b743dc5

See Also

colorPalette

Examples

```
set.seed(2)

# Simple example
canvas_petri(colors = colorPalette("origami"))

# Advanced example
canvas_petri(colors = "white", hole = 0.8, attractors = 5000)
```

`canvas_phyllotaxis` *Draw a Phyllotaxis*

Description

This function draws a phyllotaxis which resembles the arrangement of leaves on a plant stem.

Usage

```
canvas_phyllotaxis(
  colors,
  background = "#fafafa",
  iterations = 10000,
  angle = 137.5,
  size = 0.01,
  alpha = 1,
  p = 0.5
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the background.
`iterations`	the number of iterations of the algorithm.
`angle`	the angle at which to place the artwork.
`size`	the size of the lines.
`alpha`	transparency of the points.
`p`	probability of drawing a point on each iteration.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Phyllotaxis

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_phyllotaxis(colors = colorPalette("tuscany1"))
```

---

canvas_planet *Draw Planets*

Description

This function paints one or multiple planets and uses a cellular automata to fill their surfaces.

Usage

```
canvas_planet(
  colors,
  threshold = 4,
  iterations = 200,
  starprob = 0.01,
  fade = 0.2,
  radius = NULL,
  center.x = NULL,
  center.y = NULL,
  light.right = TRUE,
  resolution = 1500
)
```

Arguments

colors	a character specifying the colors used for a single planet. Can also be a list where each entry is a vector of colors for a planet.
threshold	a character specifying the threshold for a color take.
iterations	a positive integer specifying the number of iterations of the algorithm.
starprob	a value specifying the probability of drawing a star in outer space.
fade	a value specifying the amount of fading to apply.

radius	a numeric (vector) specifying the radius of the planet(s).
center.x	the x-axis coordinate(s) for the center(s) of the planet(s).
center.y	the y-axis coordinate(s) for the center(s) of the planet(s).
light.right	whether to draw the light from the right or the left.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://fronkonstin.com/2021/01/02/neighborhoods-experimenting-with-cyclic-cellular-automata/

Examples

```
set.seed(1)

# Simple example
canvas_planet(colors = colorPalette("lava"), threshold = 3)

# Advanced example
colors <- list(
  c("khaki1", "lightcoral", "lightsalmon"),
  c("dodgerblue", "forestgreen", "white"),
  c("gray", "darkgray", "beige")
)
canvas_planet(colors,
  radius = c(800, 400, 150),
  center.x = c(1, 500, 1100),
  center.y = c(1400, 500, 1000),
  starprob = 0.005
)
```

canvas_polylines *Draw Polygons and Lines*

Description

This function draws many points on the canvas and connects these points into a polygon. After repeating this for all the colors, the edges of all polygons are drawn on top of the artwork.

Usage

```
canvas_polylines(
  colors,
  background = "#fafafa",
  ratio = 0.5,
  iterations = 1000,
  size = 0.1,
  resolution = 500
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the lines.
ratio	a positive value specifying the width of the polygons. Larger ratios cause more overlap.
iterations	a positive integer specifying the number of iterations of the algorithm.
size	a positive value specifying the size of the borders.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_polylines(colors = colorPalette("retro1"))
```

`canvas_recaman` *Draw Recaman's Sequence*

Description

This function draws Recaman's sequence on a canvas. The algorithm takes increasingly large steps backward on the positive number line, but if it is unable to it takes a step forward.

Usage

```
canvas_recaman(
  colors,
  background = "#fafafa",
  iterations = 100,
  start = 0,
  increment = 1,
  curvature = 1,
  angle = 0,
  size = 0.1,
  closed = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the background.
`iterations`	the number of iterations of the algorithm.
`start`	the starting point of the algorithm.
`increment`	the increment of each step.
`curvature`	the curvature of each line.
`angle`	the angle at which to place the artwork.
`size`	the size of the lines.
`closed`	logical. Whether to plot a curve from the end of the sequence back to the starting point.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://mathworld.wolfram.com/RecamansSequence.html

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_recaman(colors = colorPalette("tuscany1"))
```

---

canvas_ribbons *Draw Ribbons*

Description

This function paints random ribbons and (optionally) a triangle in the middle.

Usage

```
canvas_ribbons(
  colors,
  background = "#fdf5e6",
  triangle = TRUE
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork. The number of colors determines the number of ribbons.
background	a character specifying the color of the background.
triangle	logical. Whether to draw the triangle that breaks the ribbon polygons.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_ribbons(colors = colorPalette("retro1"))
```

---

canvas_segments *Draw Segments*

Description

This function draws line segments on a canvas. The length and direction of the line segments is determined randomly.

Usage

```
canvas_segments(
  colors,
  background = "#fafafa",
  n = 250,
  p = 0.5,
  H = 0.1,
  size = 0.2
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
n	a positive integer specifying the number of line segments to draw.
p	a value specifying the probability of drawing a vertical line segment.
H	a positive value specifying the scaling factor for the line segments.
size	a positive value specifying the size of the line segments.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_segments(colors = colorPalette("dark1"))
```

---

canvas_slime *Draw A Slime Mold*

Description

This function draws the Physarum polycephalum slime mold on a canvas. The algorithm simulates particles on a two-dimensional grid that move towards areas on the grid with a high intensity.

Usage

```
canvas_slime(
  colors,
  background = "#000000",
  iterations = 2000,
  agents = 1000,
  layout = c(
     "random", "gaussian", "circle", "grid",
     "clusters", "arrows", "wave", "spiral"
  ),
  resolution = 1000
)
```

Arguments

colors	a character (vector) specifying the color(s) used for the artwork.
background	a character specifying the color used for the background.
iterations	a positive integer specifying the number of iterations of the algorithm.
agents	a positive integer specifying the number of agents to use.
layout	a character specifying the initial layout of the agents. Possible options are random (default), gaussian, circle, grid, clusters, arrows and wave.

`resolution` resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://cargocollective.com/sagejenson/physarum

https://fronkonstin.com/2020/08/11/abstractions/

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_slime(colors = colorPalette("neon1"))
```

---

canvas_smoke *Draw Rainbow Smoke*

Description

This function implements the rainbow smoke algorithm.

Usage

```
canvas_smoke(
  colors,
  init = 1,
  shape = c("bursts", "clouds"),
  algorithm = c("minimum", "average"),
  resolution = 150
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
init	an integer larger than zero and lower than or equal to resolution^2 specifying the initial number of pixels to color on the canvas.
shape	a character indicating the shape of the smoke. Possible options are burst and clouds.
algorithm	a character specifying how to select a new pixel. The default option minimum selects the pixel with the smallest color difference in a single neighbor and is relatively fast. The option average selects the pixel with the smallest average color difference in all the neighbors and is relatively slow.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

http://rainbowsmoke.hu

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_smoke(colors = "all", resolution = 500)

# Advanced example
reds <- colorRampPalette(c("red", "black"))
blues <- colorRampPalette(c("goldenrod", "navyblue"))
palette <- c(reds(100), blues(100))
canvas_smoke(colors = palette, init = 3, shape = "clouds", resolution = 500)
```

canvas_splits *Draw Split Lines*

Description

This function draws split lines.

Usage

```
canvas_splits(
  colors,
  background = "#fafafa",
  iterations = 6,
  sd = 0.2,
  lwd = 0.05,
  alpha = 0.5
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork.
background	a character specifying the color used for the background (and the hole).
iterations	a positive integer specifying the number of iterations of the algorithm.
sd	a numeric value specifying the standard deviation of the angle noise.
lwd	a numeric value specifying the width of the lines.
alpha	a numeric value specifying the transparency of the lines.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

colorPalette

Examples

```
set.seed(2)

# Simple example
canvas_splits(colors = "black", sd = 0)

# Simple example
```

```
canvas_splits(colors = colorPalette("dark2"), background = "black", sd = 1)
```

`canvas_squares` *Draw Squares and Rectangles*

Description

This function paints random squares and rectangles. It works by repeatedly cutting into the canvas at random locations and coloring the area that these cuts create.

Usage

```
canvas_squares(
  colors,
  background = "#000000",
  cuts = 50,
  ratio = 1.618,
  resolution = 200,
  noise = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the borders of the squares.
`cuts`	a positive integer specifying the number of cuts to make.
`ratio`	a value specifying the `1:1` ratio for each cut.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.
`noise`	logical. Whether to add k-nn noise to the artwork. Note that adding noise increases computation time significantly in large dimensions.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_squares(colors = colorPalette("retro2"))
```

`canvas_stripes` *Draw Stripes*

Description

This function creates a brownian motion on each row of the artwork and colors it according to the height of the motion.

Usage

```
canvas_stripes(
  colors,
  n = 300,
  H = 1,
  burnin = 1
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`n`	a positive integer specifying the length of the brownian motion (effectively the width of the artwork).
`H`	a positive value specifying the square of the standard deviation of each step in the motion.
`burnin`	a positive integer specifying the number of steps to discard before filling each row.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, `<koen-derks@hotmail.com>`

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_stripes(colors = colorPalette("random", n = 10))
```

`canvas_strokes` *Draw Strokes*

Description

This function creates an artwork that resembles paints strokes. The algorithm is based on the simple idea that each next point on the grid has a chance to take over the color of an adjacent colored point but also has a change of generating a new color.

Usage

```
canvas_strokes(
  colors,
  neighbors = 1,
  p = 0.01,
  iterations = 1,
  resolution = 500,
  side = FALSE
)
```

Arguments

`colors`	a string or character vector specifying the color(s) used for the artwork.
`neighbors`	a positive integer specifying the number of neighbors a block considers when taking over a color. More neighbors fades the artwork.
`p`	a value specifying the probability of selecting a new color at each block. A higher probability adds more noise to the artwork.
`iterations`	a positive integer specifying the number of iterations of the algorithm. More iterations generally apply more fade to the artwork.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.
`side`	logical. Whether to put the artwork on its side.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

See Also

`colorPalette`

Examples

```
set.seed(1)

# Simple example
canvas_strokes(colors = colorPalette("tuscany1"))
```

`canvas_swirls` *Draw Swirls*

Description

This function draws swirling stripes on a canvas by simulating a particle system.

Usage

```
canvas_swirls(
  colors,
  background = "#fafafa",
  iterations = 250,
  n = 250,
  curvature = 0.005,
  lwd = 0.1,
  resolution = 500
)
```

Arguments

`colors`	a character (vector) specifying the color(s) used for the artwork.
`background`	a character specifying the color used for the background.
`iterations`	a positive integer specifying the number of iterations of the algorithm.
`n`	a positive integer specifying the number of particles.
`curvature`	a positive number specifying the curvature of the lines. Larger values imply relatively curved lines, while lower values produce relatively straight lines.
`lwd`	expansion factor for the line width.
`resolution`	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://mattdesl.svbtle.com/generative-art-with-nodejs-and-canvas

See Also

colorPalette

Examples

```
set.seed(2)

# Simple example
canvas_swirls(colors = colorPalette("origami"))
```

---

canvas_tiles *Draw Portuguese Tiles*

Description

This function uses a reaction diffusion algorithm in an attempt to draw a Portuguese-styled tiling pattern.

Usage

```
canvas_tiles(
  colors,
  background = "#ffffff",
  iterations = 1000,
  size = 5,
  col.line = "#000000",
  resolution = 100,
  layout = NULL
)
```

Arguments

colors	a string or character vector specifying the color(s) used for the artwork, or a list containing a set of colors for each unique tile on the wall.
background	a character specifying the color of the background.
iterations	a positive integer specifying the number of iterations of the algorithm.
size	a positive integer specifying how many tiles should be in each row of the wall.
col.line	a character specifying the color of the joints between the tiles.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.
layout	optional. A matrix containing integers ranging from 1 to the maximum number of unique tiles (i.e., `length(colors)`) specifying the placement of the tiles on the wall.

Value

A `ggplot` object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/ReactionâĂŞdiffusion_system

See Also

`colorPalette`

Examples

```
set.seed(3)

# Simple example
canvas_tiles(colors = colorPalette("azul"), iterations = 5000)

# Advanced example
canvas_tiles(colors = list(
  colorPalette("blossom"),
  colorPalette("neon1"),
  colorPalette("dark1")
))

# Custom layout
layout <- matrix(c(
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1,
```

```
  1, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1,
  1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
  1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1,
  1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1,
  1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
), nrow = 11, byrow = TRUE)
canvas_tiles(
  colors = list(colorPalette("azul"), colorPalette("blossom")),
  size = nrow(layout), layout = layout
)

# Another custom layout
set.seed(11)
layout <- matrix(c(
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 1, 2, 2, 3, 3, 2, 2, 4, 4, 4, 2, 5, 5, 5, 2, 6, 2, 6, 2,
  2, 1, 2, 1, 2, 3, 2, 3, 2, 2, 4, 2, 2, 5, 2, 2, 2, 6, 2, 6, 2,
  2, 1, 1, 1, 2, 3, 3, 2, 2, 2, 4, 2, 2, 2, 5, 2, 2, 2, 6, 2, 2,
  2, 1, 2, 1, 2, 3, 2, 3, 2, 2, 4, 2, 5, 5, 5, 2, 2, 2, 6, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
), nrow = 21, byrow = TRUE)
canvas_tiles(
  colors = list(
    colorPalette("blossom"),
    colorPalette("azul"),
    colorPalette("neon1"),
    colorPalette("mixer4"),
    colorPalette("neon2"),
    colorPalette("vrolik1"),
    colorPalette("blackwhite")
  ),
  iterations = 2000,
  size = nrow(layout), layout = layout
)
```

canvas_turmite *Draw Turmites*

Description

This function paints a turmite. A turmite is a Turing machine which has an orientation in addition to a current state and a "tape" that consists of a two-dimensional grid of cells.

Usage

```
canvas_turmite(
  colors,
  background = "#fafafa",
  p = 0,
  iterations = 1000000,
  resolution = 500,
  noise = FALSE
)
```

Arguments

colors	a character specifying the color used for the artwork. The number of colors determines the number of turmites.
background	a character specifying the color used for the background.
p	a value specifying the probability of a state switch within the turmite.
iterations	a positive integer specifying the number of iterations of the algorithm.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.
noise	logical. Whether to add k-nn noise to the artwork. Note that adding noise increases computation time significantly in large dimensions.

Details

The turmite algorithm consists of the following steps: 1) turn on the spot (left, right, up, down) 2) change the color of the square 3) move forward one square.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://en.wikipedia.org/wiki/Turmite

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_turmite(colors = colorPalette("dark2"))
```

---

canvas_watercolors *Draw Watercolors*

Description

This function paints watercolors on a canvas.

Usage

```
canvas_watercolors(
  colors,
  background = "#fafafa",
  layers = 50,
  depth = 2,
  resolution = 250
)
```

Arguments

colors	a string specifying the color used for the artwork.
background	a character specifying the color used for the background.
layers	the number of layers of each color.
depth	the maximum depth of the recursive algorithm.
resolution	resolution of the artwork in pixels per row/column. Increasing the resolution increases the quality of the artwork but also increases the computation time exponentially.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

References

https://tylerxhobbs.com/essays/2017/a-generative-approach-to-simulating-watercolor-paints

See Also

colorPalette

Examples

```
set.seed(1)

# Simple example
canvas_watercolors(colors = colorPalette("tuscany2"))
```

---

colorPalette *Color Palette Generator*

Description

This function creates a random color palette, or allows the user to select a pre-implemented palette.

Usage

```
colorPalette(
  name,
  n = NULL
)
```

Arguments

name	name of the color palette. Can be random for random colors, complement for complementing colors, divergent for equally spaced colors, or random-palette for a random palette, but can also be the name of a pre-implemented palette. See the details section for a list of pre-implemented palettes.
n	the number of colors to select from the palette. Required if name = 'random', name = 'complement', or name = 'divergent'. Otherwise, if NULL, automatically selects all colors from the chosen palette.

Details

The following color palettes are implemented:


azul
bell
blackwhite
blossom
boogy1
boogy2
boogy3
dark1
dark2
dark3
flag
flora
house
gogh
jasp
jfa
jungle
klimt
kpd
lava
origami
mixer1
mixer2
mixer3
mixer4
nature
neo1
neo2
neo3
neon1
neon2
retro1
retro2
retro3
retro4
shell1
shell2
shell3
sooph
sky
tuscany1
tuscany2
tuscany3
vrolik1
vrolik2
vrolik3
vrolik4
vrolik5

Value

A vector of colors.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

Examples

```
colorPalette("divergent", 5)
```

---

saveCanvas *Save a Canvas to an External Device*

Description

This function is a wrapper around `ggplot2::ggsave`. It provides a suggested export with square dimensions for a canvas created using the `aRtsy` package.

Usage

```
saveCanvas(plot, filename, width = 7, height = 7, dpi = 300)
```

Arguments

`plot`	a ggplot2 object to be saved.
`filename`	the filename of the export.
`width`	the width of the artwork in cm.
`height`	the height of the artwork in cm.
`dpi`	the `dpi` (dots per inch) of the file.

Value

No return value, called for saving plots.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

theme_canvas *Canvas Theme for ggplot2 Objects*

Description

Add a canvas theme to the plot. The canvas theme by default has no margins and fills any empty canvas with a background color.

Usage

```
theme_canvas(x, background = NULL, margin = 0)
```

Arguments

x	a ggplot2 object.
background	a character specifying the color used for the empty canvas.
margin	margins of the canvas.

Value

A ggplot object containing the artwork.

Author(s)

Koen Derks, <koen-derks@hotmail.com>

Index