Package ‘piecepackr’

December 19, 2023

**Encoding** UTF-8

**Type** Package

**Title** Board Game Graphics

**Version** 1.13.10

**Description** Functions to make board game graphics with the 'ggplot2', 'grid', 'rayrender', 'rayvertex', and 'rgl' packages. Specializes in game diagrams, animations, and "Print & Play" layouts for the 'piecepack' <https://www.ludism.org/ppwiki> but can make graphics for other board game systems. Includes configurations for several public domain game systems such as checkers, (double-18) dominoes, go, 'piecepack', playing cards, etc.

**License** MIT + file LICENSE

**URL** [https://trevorldavis.com/piecepackr/](https://trevorldavis.com/piecepackr/) (blog),
[https://trevorldavis.com/R/piecepackr/](https://trevorldavis.com/R/piecepackr/) (pkgdown),
[https://groups.google.com/forum/#!forum/piecepackr](https://groups.google.com/forum/#!forum/piecepackr) (forum)

**BugReports** [https://github.com/piecepackr/piecepackr/issues](https://github.com/piecepackr/piecepackr/issues)

**LazyData** true

**LazyLoad** yes

**Imports** grid, gridGeometry, grImport2, grDevices, purrr, jpeg, png,
R6, rlang, stringr, tibble, tools, utils

**Suggests** animation (>= 2.7), ggplot2, gifski, gridpattern, magick,
pdf tools, rayrender (>= 0.28.8), rayvertex (>= 0.10.4), readobj
(>= 0.4.0), rgl (>= 0.106.8), scales (>= 0.5.0), systemfonts,
testthat, tweenr, vdiffr, xmpdf (>= 0.1.1), XML (>= 3.99-0.9)

**RoxygenNote** 7.2.3

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Trevor L Davis [aut, cre] (<https://orcid.org/0000-0001-6341-4639>),
Linux Foundation [dtc] (Uses some data from the "SPDX License List"
<https://github.com/spdx/license-list-XML>),
Delapouite <https://delapouite.com/> [ill] (Meeple shape extracted from
"Meeple icon"<https://game-icons.net/1x1/delapouite/meeple.html>)/
``CC BY 3.0'' (<https://creativecommons.org/licenses/by/3.0/>),
Creative Commons [ill] (`save_print_and_play()` uses ``license badges"
from Creative Commons to describe the generated print-and-play
file's license)

Maintainer  Trevor L Davis <trevor.l.davis@gmail.com>
Depends  R (>= 2.10)
Repository  CRAN
Date/Publication  2023-12-19 04:40:02 UTC

R topics documented:

- piecepackr-package  ........................................ 3
- aabb_piece  ................................................ 4
- AA_to_R  ................................................... 5
- animate_piece  ............................................. 7
- basicPieceGrobs  ............................................ 9
- font_utils  ................................................ 10
- game_systems  ............................................. 11
- geom_piece  ................................................ 15
- grid.cropmark  .............................................. 17
- grid.piece  ................................................ 20
- op_transform  .............................................. 23
- piece  ....................................................... 24
- piece3d  .................................................... 26
- piecepackr-defunct  ....................................... 28
- piece_mesh  ............................................... 29
- pmap_piece  ............................................... 31
- pp_cfg  .................................................... 32
- pp_shape  .................................................. 35
- pp_utils  .................................................. 38
- render_piece  ............................................... 39
- save_ellipsoid_obj  ....................................... 41
- save_piece_images  ....................................... 43
- save_piece_obj  ........................................... 44
- save_print_and_play  .................................... 45
- scale_x_piece  ............................................. 47
- spdx_license_list  ....................................... 49

Index  50
Description

Functions to make board game graphics with the `ggplot2`, `grid`, `rayrender`, `rayvertex`, and `rgl` packages. Specializes in game diagrams, animations, and "Print & Play" layouts for the 'piecepack' [https://www.ludism.org/ppwiki] but can make graphics for other board game systems. Includes configurations for several public domain game systems such as checkers, (double-18) dominoes, go, 'piecepack', playing cards, etc.

Package options

The following `piecepackr` function arguments may be set globally via `base::options()`:

- `piecepackr_at_inform` If `FALSE` turns off messages when affine transformation support not detected
- `piecepackr_cfg` Sets a new default for the `cfg` argument
- `piecepackr_check_cairo` If `FALSE` don’t check the version of cairo
- `piecepackr_default_units` Sets a new default for the `default.units` argument
- `piecepackr_envir` Sets a new default for the `envir` argument
- `piecepackr_metadata_inform` If `FALSE` turns off messages when support for embedding metadata not detected.
- `piecepackr_op_angle` Sets a new default for the `op_angle` argument
- `piecepackr_op_scale` Sets a new default for the `op_scale` argument
- `piecepackr_trans` Sets a new default for the `trans` argument

Author(s)

Maintainer: Trevor L Davis <trevor.l.davis@gmail.com> (ORCID)

Other contributors:

- Linux Foundation (Uses some data from the "SPDX License List" <https://github.com/spdx/license-list-XML>) [data contributor]
- Delapouite <https://delapouite.com/> (Meeple shape extracted from "Meeple icon" <https://game-icons.net/1x1/delapouite/meeple.html> / "CC BY 3.0" <https://creativecommons.org/licenses/by/3.0/> [illustrator]
- Creative Commons (‘save_print_and_play()’ uses "license badges" from Creative Commons to describe the generated print-and-play file’s license) [illustrator]
See Also

Useful links:

- blog: https://trevorldavis.com/piecepackr/
- pkgdown: https://trevorldavis.com/R/piecepackr/
- forum: https://groups.google.com/forum/#!forum/piecepackr
- Report bugs: https://github.com/piecepackr/piecepackr/issues

---

**aabb_piece**

*Calculate axis-aligned bounding box for set of game pieces*

**Description**

Calculate axis-aligned bounding box (AABB) for set of game pieces with and without an “oblique projection”.

**Usage**

```r
aabb_piece(
  df,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  envir = getOption("piecepackr.envir"),
  op_scale = getOption("piecepackr.op_scale", 0),
  op_angle = getOption("piecepackr.op_angle", 45),
  ...
)
```

**Arguments**

- `df` A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.
- `cfg` Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- `envir` Environment (or named list) containing configuration list(s).
- `op_scale` How much to scale the depth of the piece in the oblique projection (viewed from the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5 is the most common scale used in the “cabinet” projection, and 1.0 is the scale used in the “cavalier” projection.
- `op_angle` What is the angle of the oblique projection? Has no effect if op_scale is 0.
- `...` Ignored

**Details**

The “oblique projection” of a set of \((x, y, z)\) points onto the xy-plane is \((x + \lambda \times z \times \cos(\alpha), y + \lambda \times z \times \sin(\alpha))\) where \(\lambda\) is the scale factor and \(\alpha\) is the angle.
**Value**

A named list of ranges with five named elements `x`, `y`, and `z` for the axis-aligned bounding cube in `xyz`-space plus `x_op` and `y_op` for the axis-aligned bounding box of the “oblique projection” onto the `xy` plane.

**Examples**

```r
df_tiles <- data.frame(piece_side="tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1), suit=NA, angle=NA, z=NA, stringsAsFactors=FALSE)
df_coins <- data.frame(piece_side="coin_back", x=rep(4:1, 4), y=rep(4:1, each=4), suit=1:16%%2+rep(c(1,3), each=8), angle=rep(c(180,0), each=8), z=1/4+1/16, stringsAsFactors=FALSE)
df <- rbind(df_tiles, df_coins)
aabb_piece(df, op_scale = 0)
aabb_piece(df, op_scale = 1, op_angle = 45)
aabb_piece(df, op_scale = 1, op_angle = -90)
```

---

**AA_to_R**

*Helper functions for making geometric calculations.*

**Description**

to_x, to_y, to_r, to_t convert between polar coordinates (in degrees) and Cartesian coordinates. to_degrees and to_radians converts between degrees and radians. `AA_to_R` and `R_to_AA` convert back and forth between (post-multiplied) rotation matrix and axis-angle representations of 3D rotations. `R_x`, `R_y`, and `R_z` build (post-multiplied) rotation matrices for simple rotations around the x, y, and z axes.

**Usage**

```r
AA_to_R(angle = 0, axis_x = 0, axis_y = 0, axis_z = NA, ...)
R_to_AA(R = diag(3))
R_x(angle = 0)
R_y(angle = 0)
R_z(angle = 0)
to_radians(t)
to_degrees(t)
to_x(t, r)
```
to_y(t, r)
to_r(x, y)
to_t(x, y)

Arguments

angle Angle in degrees (counter-clockwise)
axis_x First coordinate of the axis unit vector.
axis_y Second coordinate of the axis unit vector.
axis_z Third coordinate of the axis unit vector (usually inferred).
... Ignored
R 3D rotation matrix (post-multiplied)
t Angle in degrees (counter-clockwise)
r Radial distance
x Cartesian x coordinate
y Cartesian y coordinate

Details

pp_cfg uses polar coordinates to determine where the "primary" and "directional" symbols are located on a game piece. They are also useful for drawing certain shapes and for making game diagrams on hex boards.

piecepackr and grid functions use angles in degrees but the base trigonometry functions usually use radians.

piecepackr's 3D graphics functions save_piece_obj, piece, and piece3d use the axis-angle representation for 3D rotations. The axis-angle representation involves specifying a unit vector indicating the direction of an axis of rotation and an angle describing the (counter-clockwise) rotation around that axis. Because it is a unit vector one only needs to specify the first two elements, axis_x and axis_y, and we are able to infer the 3rd element axis_z. The default of axis = 0, axis_y = 0, and implied axis_z = 1 corresponds to a rotation around the z-axis which is reverse-compatible with the originally 2D angle interpretation in grid.piece. In order to figure out the appropriate axis-angle representation parameters R_to_AA, R_x, R_y, and R_z allow one to first come up with an appropriate (post-multiplied) 3D rotation matrix by chaining simple rotations and then convert them to the corresponding axis-angle representation. Pieces are rotated as if their center was at the origin.

See Also

animate_piece

Examples

```
to_x(90, 1)
to_y(180, 0.5)
to_t(0, -1)
to_r(0.5, 0)
all.equal(pi, to_radians(to_degrees(pi)))
```

# default axis-angle axis is equivalent to a rotation about the z-axis
```
all.equal(AA_to_R(angle=60), R_z(angle=60))
```

# axis-angle representation of 90 rotation about the x-axis
```
R_to_AA(R_x(90))
```

# find Axis-Angle representation of first rotating about x-axis 180 degrees
# and then rotating about z-axis 45 degrees
```
R_to_AA(R_x(180) %*% R_z(45))
```

---

animate_piece  

Animate board game pieces

Description

animate_piece() animates board game pieces.

Usage

```
animate_piece(
  dfs,
  file = "animation.gif",
  ..., 
  annotate = TRUE, 
  .f = piecepackr::grid.piece, 
  cfg = getOption("piecepackr.cfg", NULL), 
  envir = getOption("piecepackr.envir", game_systems("sans")), 
  n_transitions = 0L, 
  n pauses = 1L, 
  fps = n_transitions + n_pauses, 
  width = NULL, 
  height = NULL, 
  ppi = NULL, 
  new_device = TRUE, 
  annotation_scale = NULL
)
```

Arguments

dfs  

A list of data frames of game data to plot.

file  

Filename to save animation unless NULL in which case it uses the current graphics device.
Arguments to `pmap_piece`

annotate  If TRUE or "algebraic" annotate the plot with "algebraic" coordinates, if FALSE or "none" don't annotate, if "cartesian" annotate the plot with "cartesian" coordinates.

.f  Low level graphics function to use e.g. `grid.piece()`, `piece3d()`, `piece()`, or `piece_mesh()`.

cfg  A piecepackr configuration list

envir  Environment (or named list) of piecepackr configuration lists

n_transitions  Integer, if over zero (the default) how many transition frames to add between moves.

n_pauses  Integer, how many paused frames per completed move.

fps  Double, frames per second.

width  Width of animation (in inches). Inferred by default.

height  Height of animation (in inches). Inferred by default.

ppi  Resolution of animation in pixels per inch. By default set so image max 600 pixels wide or tall.

new_device  If file is NULL should we open up a new graphics device?

annotation_scale  Multiplicative factor that scales (stretches) any annotation coordinates. By default uses `attr(df, "scale_factor") %||% 1`.

Value

Nothing, as a side effect creates an animation.

Examples

# Basic tic-tac-toe animation
dfs <- list()
d.frame <- function(piece_side = "bit_back", ..., rank = 1L) {
  data.frame(piece_side = piece_side, ..., rank = rank,
              cfg = "checkers1", stringsAsFactors = FALSE)
}
df <- d.frame("board_back", suit = 2L, rank = 3L, x = 2, y = 2, id = "1")
dfs[[4L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 2, y = 2, id = "2"))
dfs[[2L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 1, y = 2, id = "3"))
dfs[[3L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 3, y = 1, id = "4"))
dfs[[4L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 1, y = 3, id = "5"))
dfs[[5L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 1, y = 1, id = "6"))
dfs[[6L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 3, y = 3, id = "7"))
dfs[[7L]] <- df
basicPieceGrobs

Description

basicPieceGrob is the most common "grob" function that grid.piece uses to create grid graphical grob objects. picturePieceGrobFn is a function that returns a "grob" function that imports graphics from files found in its directory argument.

Usage

basicPieceGrob(piece_side, suit, rank, cfg = pp_cfg())

picturePieceGrobFn(directory, filename_fn = find_pp_file)

pyramidTopGrob(piece_side, suit, rank, cfg = pp_cfg())

previewLayoutGrob(piece_side, suit, rank, cfg = pp_cfg())

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>piece_side</td>
<td>A string with piece and side separated by a underscore e.g. &quot;coin_face&quot;</td>
</tr>
<tr>
<td>suit</td>
<td>Number of suit (starting from 1).</td>
</tr>
<tr>
<td>rank</td>
<td>Number of rank (starting from 1)</td>
</tr>
<tr>
<td>cfg</td>
<td>Piecepack configuration list or pp_cfg object.</td>
</tr>
<tr>
<td>directory</td>
<td>Directory that picturePieceGrobFn will look in for piece graphics.</td>
</tr>
</tbody>
</table>
filename_fn  Function that takes arguments directory, piece_side, suit, rank, and optionally cfg and returns the (full path) filename of the image that the function returned by picturePieceGrobFn should import.

Examples

```r
if (requireNamespace("grid", quietly = TRUE) && piecepackr:::device_supports_unicode()) {
  cfg <- pp_cfg(list(grob_fn.tile=basicPieceGrob, invert_colors=TRUE))
  grid.piece("tile_face", suit=1, rank=3, cfg=cfg)
}

# May take more than 5 seconds on CRAN servers
try({
  if (requireNamespace("grid", quietly = TRUE) && capabilities(c("cairo"))) {
    cfg <- pp_cfg(list(grob_fn.tile=basicPieceGrob, invert_colors=TRUE))
    directory <- tempdir()
    save_piece_images(cfg, directory=directory, format="svg", angle=0)
    cfg2 <- pp_cfg(list(grob_fn=picturePieceGrobFn(directory)))
    grid::grid.newpage()
    grid.piece("coin_back", suit=3, rank=5, cfg=cfg2)
  }
})
```

---

font_utils  

Font utility functions

Description

get_embedded_font() returns which font is actually embedded by cairo_pdf() for a given character. has_font() tries to determine if a given font is available on the OS.

Usage

get_embedded_font(font, char)

has_font(font)

Arguments

- **font**  A character vector of font(s).
- **char**  A character vector of character(s) to be embedded by grid::grid.text()

Details

get_embedded_font() depends on the suggested pdftools package being installed and R being compiled with Cairo support. has_font() depends on either the suggested systemfonts (preferred) or pdftools packages being installed.
Value

get_embedded_font() returns character vector of fonts that were actually embedded by cairo_pdf(). NA’s means no embedded font detected: this either means that no font was found or that a color emoji font was found and instead of a font an image was embedded.

Examples

if (requireNamespace("pdftools", quietly = TRUE) &&
    capabilities("cairo") &&
    !piecepackr:::is_cairo_maybe_buggy()) {
  chars <- c("a", "\u2666")
  fonts <- c("sans", "Sans Noto", "Noto Sans", "Noto Sans Symbols2")
  try(get_embedded_font(fonts, chars))
}

if (requireNamespace("systemfonts", quietly = TRUE) ||
    (requireNamespace("pdftools", quietly = TRUE) &&
    capabilities("cairo") &&
    !piecepackr:::is_cairo_maybe_buggy())) {
  try(has_font("Dejavu Sans"))
}

Description

game_systems returns a list of pp_cfg objects representing several game systems and pieces. to_subpack and to_hexpack will attempt to generate matching (piecepack stackpack) subpack and (piecepack) hexpack pp_cfg R6 objects respectively given a piecepack configuration.

Usage

game_systems(style = NULL, round = FALSE, pawn = "token")

to_hexpack(cfg = getOption("piecepackr.cfg", pp_cfg()))

to_subpack(cfg = getOption("piecepackr.cfg", pp_cfg()))

Arguments

style If NULL (the default) uses suit glyphs from the default “sans” font. If "dejavu" it will use suit glyphs from the "DejaVu Sans" font (must be installed on the system).

round If TRUE the “shape” of “tiles” and “cards” will be “roundrect” instead of “rect” (the default).
If "token" (default) the piecepack pawn will be a two-sided token in a “halma” outline, if "peg-doll" the piecepack pawn will be a “peg doll” style pawn, and if "joystick" the piecepack pawn will be a “joystick” style pawn. Note for the latter two pawn styles only pawn_top will work with grid.piece.

cfg
List of configuration options

Details
Contains the following game systems:

alquerque Boards and pieces in six color schemes for Alquerque

checkers1, checkers2 Checkers and checkered boards in six color schemes. Checkers are represented by a piecepackr “bit”. The “board” “face” is a checkered board and the “back” is a lined board. Color is controlled by suit and number of rows/columns by rank. checkers1 has one inch squares and checkers2 has two inch squares.

chess1, chess2 Chess pieces, boards, and dice in six color schemes. Chess pieces are represented by a “bit” (face). The “board” “face” is a checkered board and the “back” is a lined board. Color is controlled by suit and number of rows/columns by rank. chess1 has one inch squares and chess2 has two inch squares. Currently uses print-and-play style discs instead of 3D Staunton chess pieces.

dice Traditional six-sided piped dice in six color schemes (color controlled by their suit).

dice_d4, dice_numeral, dice_d8, dice_d10, dice_d10_percentile, dice_d12, dice_d20 Polyhedral dice most commonly used to play wargames, roleplaying games, and trading card games:

dice_d4 Four-sided dice in six color schemes (color controlled by their suit). Tetrahedrons with the rank as a numeral at the top point.

dice_numeral Six-sided dice with numerals instead of pips in six color schemes (color controlled by their suit).

dice_d8 Eight-sided dice in six color schemes (color controlled by their suit). Octahedrons with the rank as a numeral at the top face.

dice_d10 Ten-sided dice in six color schemes (color controlled by their suit). Pentagonal trapezohedrons with the rank as a numeral at the top face. The rank of ten is represented by a zero.

dice_d10_percentile Ten-sided dice in six color schemes (color controlled by their suit). Pentagonal trapezohedrons with the rank as a numeral followed by a zero at the top face. The rank of ten is represented by a zero.

dice_d12 Twelve-sided dice in six color schemes (color controlled by their suit). Dodecahedrons with the rank as a numeral at the top face.

dice_d20 Twenty-sided dice in six color schemes (color controlled by their suit). Icosahedrons with the rank as a numeral at the top face.

dice_fudge “Fudge” dice in six color schemes (color controlled by their suit). “Fudge” dice have three ranks “+”, “”, and “.” repeated twice.

dominoes, dominoes_black, dominoes_blue, dominoes_green, dominoes_red, dominoes_white, dominoes_yellow Traditional pipped dominoes in six color schemes (dominoes and dominoes_white are the same). In each color scheme the number of pips on the “top” of the domino is controlled by their “rank” and on the “bottom” by their “suit”. Supports up to double-18 sets.
**dominoes_chinese, dominoes_chinese_black**  
*dominoes_chinese* has Asian-style six-sided pipped dice with white background and black and red pips. The “tile”'s are Chinese dominoes (1” x 2.5”) whose number of pips are controlled by both their “rank” and their “suit”. *dominoes_chinese_black* are like *dominoes_chinese* but the dice and dominoes have a black background and white and red pips.

**go**  
Go stones and lined boards in six color schemes. Go stones are represented by a “bit” and the board is a “board”. Color is controlled by suit and number of rows/columns by rank.

**meeples**  
Various morris aka mills aka merels games in six colors. Color is controlled by suit and “size” of morris board is controlled by rank e.g. “Six men’s morris” corresponds to a rank of 6 and “Nine men’s morris” corresponds to a rank of 9. Game pieces are represented by stones.

**piecepack, dual_piecepacks_expansion, playing_cards_expansion, hexpack, subpack, piecepack_inverted**  
The piecepack is a public domain game system invented by James "Kyle" Droscha. See [https://www.ludism.org/ppwiki](https://www.ludism.org/ppwiki) for more info about the piecepack and its accessories/expansions.

**piecepack**  
A standard piecepack. The configuration also contains the following piecepack accessories:

- **piecepack dice cards**  
  An accessory proposed by John Braley. See [https://www.ludism.org/ppwiki/PiecepackDiceCards](https://www.ludism.org/ppwiki/PiecepackDiceCards).

- **piecepack matchsticks**  
  A public domain accessory developed by Dan Burkey. See [https://www.ludism.org/ppwiki/PiecepackMatchsticks](https://www.ludism.org/ppwiki/PiecepackMatchsticks).

- **piecepack pyramids**  
  A public domain accessory developed by Tim Schutz. See [https://www.ludism.org/ppwiki/PiecepackPyramids](https://www.ludism.org/ppwiki/PiecepackPyramids).

- **piecepack saucers**  

- **piecepack_inverted**  
The standard piecepack with its color scheme inverted. Intended to aid in highlighting special pieces in diagrams.

- **dual_piecepacks_expansion**  

- **playing_cards_expansion**  
  A piecepack with the standard “French” playing card suits. See [https://www.ludism.org/ppwiki/PlayingCardsExpansion](https://www.ludism.org/ppwiki/PlayingCardsExpansion).

- **hexpack**  
  A hexagonal extrapolation of the piecepack designed by Nathan Morse and Daniel Wilcox. See [https://boardgamegeek.com/boardgameexpansion/35424/hexpack](https://boardgamegeek.com/boardgameexpansion/35424/hexpack).

- **subpack**  
  A mini piecepack. Designed to be used with the piecepack to make piecepack “stackpack” diagrams. See [https://www.ludism.org/ppwiki/StackPack](https://www.ludism.org/ppwiki/StackPack).

**playing_cards, playing_cards_colored, playing_cards_tarot**  
Poker-sized card components for various playing card decks:

- **playing_cards**  
  A traditional deck of playing cards with 4 suits and 13 ranks (A, 2-10, J, Q, K) plus a 14th “Joker” rank.

- **playing_cards_colored**  
  Like playing_cards but with five colored suits: red hearts, black spades, green clubs, blue diamonds, and yellow stars.

- **playing_cards_tarot**  
  A (French Bourgeois) deck of tarot playing cards: first four suits are hearts, spades, clubs, and diamonds with 14 ranks (ace through jack, knight, queen, king) plus a 15th “Joker” rank and a fifth “suit” of 22 trump cards (1-21 plus an “excuse”).
**reversi** Boards and pieces for Reversi. "board_face" provides lined boards with colored backgrounds. "board_back" provides checkered boards. "bit_face" / "bit_back" provides circular game tokens with differently colored sides: red paired with green, black paired with white, and blue paired with yellow.

See Also

pp_cfg for information about the pp_cfg objects returned by game_systems.

Examples

cfgs <- game_systems(pawn = "joystick")
names(cfgs)

# May take more than 5 seconds on CRAN servers
# standard dice, meeples, and joystick pawns
if (requireNamespace("grid", quietly = TRUE) & piecepackr:::device_supports_unicode()) {
  opt <- options(piecepackr.at.inform = FALSE)
  grid::grid.newpage()
  dice <- c("d4", "numeral", "d8", "d10", "d12", "d20")
  cfg <- paste0("dice_", dice)
  grid.piece("die_face", suit = c(1:6, 1), rank = 1:6,
            cfg = cfg, envir = cfgs, x = 1:6, y = 1,
            default.units = "in", op_scale = 0.5)
  grid.piece("die_face", rank=1:6, suit=1:6,
            x=1:6, y=2, default.units="in",
            op_scale=0.5, cfg=cfgs$dice)
  grid.piece("bit_face", suit=1:6,
            x=1:6, y=3, default.units="in",
            op_scale=0.5, cfg=cfgs$meeple)
  grid.piece("pawn_top", suit=1:6,
            x=1:6, y=4, default.units="in",
            op_scale=0.5, cfg=cfgs$piecepack)
  options(opt)
}

# dominoes
if (requireNamespace("grid", quietly = TRUE)) {
  grid::grid.newpage()
  colors <- c("black", "red", "green", "blue", "yellow", "white")
  cfg <- paste0("dominoes_", rep(colors, 2))
  grid.piece("tile_face", suit=1:12, rank=1:12+1,
            cfg=cfg, envir=cfgs,
            x=rep(6:1, 2), y=rep(2*2:1, each=6),
            default.units="in", op_scale=0.5)
}

# piecepack "playing card expansion"
if (requireNamespace("grid", quietly = TRUE) & piecepackr:::device_supports_unicode()) {
  grid::grid.newpage()
  df_tiles <- data.frame(piece_side="tile_back",
                         x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1),
                         suit=NA, angle=NA, z=1/8,

df_coins <- data.frame(piece_side="coin_back", 
                      x=rep(4:1, 4), y=rep(4:1, each=4), 
                      suit=c(1,4,1,4,1,4,1,4,1,2,3,2,3,2,3,2), 
                      angle=rep(c(180,0), each=8), z=1/4+1/16, 
                      stringsAsFactors=FALSE)

df <- rbind(df_tiles, df_coins)

pmap_piece(df, cfg = cfgs$playing_cards_expansion, op_scale=0.5, 
            default.units="in")

---

**geom_piece**

*Draw board game pieces with ggplot2*

### Description

`geom_piece()` creates a ggplot2 geom. `aes_piece()` takes a data frame and generates an appropriate ggplot2::aes() mapping.

### Usage

```r
geom_piece(
  mapping = NULL, 
  data = NULL, 
  stat = "identity", 
  position = "identity", 
  ...,
  envir = getOption("piecepackr.envir", piecepackr::game_systems()),
  op_scale = getOption("piecepackr.op_scale", 0),
  op_angle = getOption("piecepackr.op_angle", 45),
  inherit.aes = TRUE
)

aes_piece(df)
```

### Arguments

- **mapping**
  - Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**
  - The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

**stat**

The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")

**position**

Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

... Aesthetics, used to set an aesthetic to a fixed value.

**envir**

Environment (or named list) containing configuration list(s).

**op_scale**

How much to scale the depth of the piece in the oblique projection (viewed from the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5 is the most common scale used in the “cabinet” projection, and 1.0 is the scale used in the “cavalier” projection.

**op_angle**

What is the angle of the oblique projection? Has no effect if op_scale is 0.

**inherit.aes**

If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

**df**

A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.

**Details**

gem.piece() requires a fixed scale coordinate system with an aspect ratio of 1 as provided by ggplot2::coord_fixed(). geom.piece() also requires that cfg is a character vector (and not a pp_cfg() object). In particular if using op_transform() one should set its argument cfg_class = "character" if intending for use with geom.piece().

**Aesthetics**

gem.piece() understands the following aesthetics (required aesthetics are in bold). See pieceGrob() for more details.

- x
- y
- z
- piece_side
- rank
- suit
- cfg
- width
- height
- depth
grid.cropmark

Crop Mark Grob

Description

grid.cropmark() draws “crop marks” to the active graphics device. cropmarkGrob() is its grid grob counterpart. Intended for use in adding crop marks around game pieces in print-and-play layouts.
Usage

cropmarkGrob(
  ..., 
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  x = unit(0.5, "npc"),
  y = unit(0.5, "npc"),
  angle = 0,
  width = NA,
  height = NA,
  scale = 1,
  default.units = "npc",
  envir = getOption("piecepackr.envir"),
  name = NULL,
  gp = NULL,
  vp = NULL,
  bleed = unit(0.125, "in"),
  cm_select = "12345678",
  cm_width = unit(0.25, "mm"),
  cm_length = unit(0.125, "in")
)

grid.cropmark(..., draw = TRUE)

Arguments

... cropmarkGrob() ignores; grid.cropmark() passes to cropmarkGrob().
piece_side A string with piece and side separated by a underscore e.g. "coin_face"
suit Number of suit (starting from 1).
rank Number of rank (starting from 1)
cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base:::dynGet().
x Where to place piece on x axis of viewport
y Where to place piece on y axis of viewport
angle Angle (on xy plane) to draw piece at
width Width of piece
height Height of piece
scale Multiplicative scaling factor to apply to width, height, and depth.
default.units A string indicating the default units to use if 'x', 'y', 'width', and/or 'height' are only given as numeric vectors.
envir Environment (or named list) containing configuration list(s).
name  A character identifier (for grid)
gp    An object of class “gpar”.
vp    A grid viewport object (or NULL).
bleed Bleed zone size to assume:
       • If bleed is a grid::unit() simply use it
       • If bleed is numeric then convert via grid::unit(bleed, default.units)
       • If bleed is TRUE assume 1/8 inch bleed zone size
       • If bleed is FALSE assume 0 inch bleed zone size

cm_select A string of integers from "1" to "8" indicating which crop marks to draw. "1"
            represents the top right crop mark then we proceeding clockwise to "8" which
            represents the top left crop mark. Default "12345678" draws all eight crop
            marks.

cm_width  Width of crop mark.

cm_length Length of crop mark.
draw  A logical value indicating whether graphics output should be produced.

Value
A grid grob.

Examples
if (requireNamespace("grid", quietly = TRUE) && piecepackr:::device_supports_unicode()) {
  cfg <- pp_cfg(list(mat_color = "pink", mat_width=0.05, border_color=NA))
  grid::grid.newpage()
  df <- data.frame(piece_side = "tile_face", suit = 2, rank = 2,
       x = 2, y = 2, angle = 0,
       stringsAsFactors = FALSE)
  pmap_piece(df, grid.cropmark, cfg = cfg, default.units = "in")
  pmap_piece(df, grid.piece, cfg = cfg, default.units = "in", bleed=TRUE)
} if (requireNamespace("grid", quietly = TRUE) && piecepackr:::device_supports_unicode()) {
  grid::grid.newpage()
  df <- data.frame(piece_side = "coin_back", suit = 2, rank = 2,
       x = 2, y = 2, angle = 0,
       stringsAsFactors = FALSE)
  pmap_piece(df, grid.cropmark, cfg = cfg, default.units = "in", bleed=TRUE)
  pmap_piece(df, grid.piece, cfg = cfg, default.units = "in", bleed=TRUE)
}
grid.piece  
\hspace{1cm} \textit{Draw board game pieces with grid}

\textbf{Description}

\texttt{grid.piece()} draws board game pieces onto the graphics device. \texttt{pieceGrob()} is its grid “grob” counterpart.

\textbf{Usage}

\begin{verbatim}
    pieceGrob(
        piece_side = "tile_back",
        suit = NA,
        rank = NA,
        cfg = getOption("piecepackr.cfg", pp_cfg()),
        x = unit(0.5, "npc"),
        y = unit(0.5, "npc"),
        z = NA,
        angle = 0,
        ..., 
        width = NA,
        height = NA,
        depth = NA,
        op_scale = getOption("piecepackr.op_scale", 0),
        op_angle = getOption("piecepackr.op_angle", 45),
        default.units = getOption("piecepackr.default.units", "npc"),
        envir = getOption("piecepackr.envir"),
        name = NULL,
        gp = NULL,
        vp = NULL,
        scale = 1,
        alpha = 1,
        type = "normal",
        bleed = FALSE
    )

    grid.piece(
        piece_side = "tile_back",
        suit = NA,
        rank = NA,
        cfg = getOption("piecepackr.cfg", pp_cfg()),
        x = unit(0.5, "npc"),
        y = unit(0.5, "npc"),
        z = NA,
        angle = 0,
        ..., 
        width = NA,
    )
\end{verbatim}
grid.piece

height = NA,
depth = NA,
op_scale =getOption("piecepackr.op_scale", 0),
op_angle =getOption("piecepackr.op_angle", 45),
default.units =getOption("piecepackr.default.units", "npc"),
envir =getOption("piecepackr.envir"),
name = NULL,
gp = NULL,
draw = TRUE,
vp = NULL,
scale = 1,
alpha = 1,
type = "normal",
bleed = FALSE
)

Arguments

piece_side A string with piece and side separated by a underscore e.g. "coin_face"
suit Number of suit (starting from 1).
rank Number of rank (starting from 1)
cfg Piecepack configuration list or pp_CFG object, a list of pp_CFG objects, or a
c character vector referring to names in envir or a character vector referring to
object names that can be retrieved by base::dynGet().
x Where to place piece on x axis of viewport
y Where to place piece on y axis of viewport
z z-coordinate of the piece. Has no effect if op_scale is 0.
angle Angle (on xy plane) to draw piece at
... Ignored.
width Width of piece
height Height of piece
depth Depth (thickness) of piece. Has no effect if op_scale is 0.
op_scale How much to scale the depth of the piece in the oblique projection (viewed from
the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5
is the most common scale used in the “cabinet” projection, and 1.0 is the scale
used in the “cavalier” projection.
op_angle What is the angle of the oblique projection? Has no effect if op_scale is 0.
default.units A string indicating the default units to use if 'x', 'y', 'width', and/or 'height' are
only given as numeric vectors.
envir Environment (or named list) containing configuration list(s).
name A character identifier (for grid)
gp An object of class “gpar”.
vp A grid viewport object (or NULL).
grid.piece

- **scale**: Multiplicative scaling factor to apply to width, height, and depth.
- **alpha**: Alpha channel for transparency.
- **type**: Type of grid grob to use. Either "normal" (default), "picture", "raster", or "transformation". "picture" exports to (temporary) svg and re-imports as a grImport2::pictureGrob. "raster" exports to (temporary) png and re-imports as a grid::rasterGrob. "transformation" uses the affine transformation feature only supported in R 4.2+ within select graphic devices. The latter three can be useful if drawing pieces really big or small and don't want to mess with re-configuring fontsizes and linewidths.
- **bleed**: If FALSE do not add a “bleed” zone around the piece, otherwise add a “bleed” zone around the piece:
  - If bleed is TRUE we will add 1/8 inch bleeds
  - If bleed is a grid::unit() we will use it as bleed size
  - If bleed is numeric we will convert to grid::unit() via grid::unit(bleed, default.units)

A non-FALSE bleed is incompatible with op_scale > 0 (drawing in an “oblique projection”).
- **draw**: A logical value indicating whether graphics output should be produced.

**Value**

A grid grob object. If draw is TRUE then as a side effect grid.piece() will also draw it to the graphics device.

**See Also**

pmap_piece() which applies pieceGrob() over rows of a data frame.

**Examples**

```r
if (requireNamespace("grid", quietly = TRUE) && piecepackr:::device_supports_unicode()) {
  opt <- options(piecepackr.at.inform = FALSE)
  on.exit(options(opt))

  draw_pp_diagram <- function(cfg=pp_cfg(), op_scale=0) {
    g.p <- function(...) {
      grid.piece(..., op_scale=op_scale, cfg=cfg, default.units="in")
    }
    g.p("tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1))
    g.p("tile_back", x=0.5+3, y=0.5+1, z=1/4+1/8)
    g.p("tile_back", x=0.5+3, y=0.5+1, z=2/4+1/8)
    g.p("die_face", suit=3, rank=5, x=1, y=1, z=1/4+1/4)
    g.p("pawn_face", x=1, y=4, z=1/4+1/2, angle=90)
    g.p("coin_back", x=3, y=4, z=1/4+1/16, angle=180)
    g.p("coin_back", suit=4, x=3, y=4, z=1/4+1/8+1/16, angle=180)
    g.p("coin_back", suit=2, x=3, y=1, z=3/4+1/8, angle=90)
  }
}

# default piecepack, orthogonal projection
```
op_transform

Oblique projection helper function

Description

Guesses z coordinates and sorting order to more easily make 3D graphics with pmap_piece.

Usage

op_transform(
  df,
  ...
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  envir = getOption("piecepackr.envir"),
  op_angle = getOption("piecepackr.op_angle", 45),
  pt_thickness = 0.01,
  as_top = character(0),
  cfg_class = "list"
)

Arguments

df A data frame with coordinates and dimensions in inches

... Ignored
cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector of pp_cfg objects

envir Environment (or named list) containing configuration list(s).
op_angle  Intended oblique projection angle (used for re-sorting)
pt_thickness  Thickness of pyramid tip i.e. value to add to the z-value of a pyramid top if it is a (weakly) smaller ranked pyramid (top) placed on top of a larger ranked pyramid (top).
as_top  Character vector of components whose "side" should be converted to "top" e.g. c("pawn_face").
cfg_class  Either "list" (default) or "character". Desired class of the cfg column in the returned tibble. "list" is more efficient for use with pmap_piece() but geom_piece() needs "character".

Details

The heuristics used to generate guesses for z coordinates and sorting order aren’t guaranteed to work in every case. In some cases you may get better sorting results by changing the op_angle or the dimensions of pieces.

Value

A tibble with extra columns added and re-sorted rows

See Also


Examples

df <- tibble::tibble(piece_side="tile_back",
x=c(2,2,2,4,6,6,4,2,5),
y=c(4,4,4,4,4,2,2,2,3))
cfg <- game_systems()$piecepack
pmap_piece(df, op_angle=135, trans=op_transform, op_scale=0.5, default.units="in", cfg=cfg)
Usage

```r
piece(
    piece_side = "tile_back",
    suit = NA,
    rank = NA,
    cfg = getOption("piecepackr.cfg", pp_cfg()),
    x = 0,
    y = 0,
    z = NA,
    angle = 0,
    axis_x = 0,
    axis_y = 0,
    width = NA,
    height = NA,
    depth = NA,
    envir = getOption("piecepackr.envir"),
    ..., 
    scale = 1,
    res = 72
)
```

Arguments

- **piece_side** A string with piece and side separated by a underscore e.g. "coin_face"
- **suit** Number of suit (starting from 1).
- **rank** Number of rank (starting from 1)
- **cfg** Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- **x** Where to place piece on x axis of viewport
- **y** Where to place piece on y axis of viewport
- **z** z-coordinate of the piece. Has no effect if op_scale is 0.
- **angle** Angle (on xy plane) to draw piece at
- **axis_x** First coordinate of the axis unit vector.
- **axis_y** Second coordinate of the axis unit vector.
- **width** Width of piece
- **height** Height of piece
- **depth** Depth (thickness) of piece. Has no effect if op_scale is 0.
- **envir** Environment (or named list) containing configuration list(s).
- **...** Ignored.
- **scale** Multiplicative scaling factor to apply to width, height, and depth.
- **res** Resolution of the faces.
Value

A rayrender object.

See Also

See https://www.rayrender.net for more information about the rayrender package. See geometry_utils for a discussion of the 3D rotation parameterization.

Examples

```r
# May take more than 5 seconds on CRAN servers
opt <- options(cores = getOption("Ncpus"))
cfg <- game_systems("sans3d")$piecepack
if (requireNamespace("rayrender", quietly = TRUE) && all(capabilities(c("cairo", "png"))) {  
  rayrender::render_scene(piece("tile_face", suit = 3, rank = 3, cfg = cfg))
}
if (requireNamespace("rayrender", quietly = TRUE) && all(capabilities(c("cairo", "png"))) {  
  rayrender::render_scene(piece("coin_back", suit = 4, rank = 2, cfg = cfg))
}
if (requireNamespace("rayrender", quietly = TRUE) && all(capabilities(c("cairo", "png"))) {  
  rayrender::render_scene(piece("pawn_face", suit = 2, cfg = cfg))
}
options(opt)
```

---

**piece3d**

*Render board game pieces with rgl*

**Description**

piece3d draws board games pieces using the rgl package.

**Usage**

```
piece3d(  
  piece_side = "tile_back",  
  suit = NA,  
  rank = NA,  
  cfg = getOption("piecepackr.cfg", pp_cfg()),  
  x = 0,  
  y = 0,  
  z = NA,  
  angle = 0,  
  axis_x = 0,  
  axis_y = 0,  
  width = NA,  
  height = NA,  
  depth = NA,  
)```

envir = getOption("piecepackr.envir"),
...,  
scale = 1,
res = 72,
alpha = 1,
lit = FALSE,
shininess = 50,
textype = NA
)

Arguments

piece_side  A string with piece and side separated by a underscore e.g. "coin_face"
suit  Number of suit (starting from 1).
rank  Number of rank (starting from 1)
cfg  Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base:::dynGet().
x  Where to place piece on x axis of viewport
y  Where to place piece on y axis of viewport
z  z-coordinate of the piece. Has no effect if op_scale is 0.
angle  Angle (on xy plane) to draw piece at
axis_x  First coordinate of the axis unit vector.
axis_y  Second coordinate of the axis unit vector.
width  Width of piece
height  Height of piece
depth  Depth (thickness) of piece. Has no effect if op_scale is 0.
envir  Environment (or named list) containing configuration list(s).
...  Ignored.
scale  Multiplicative scaling factor to apply to width, height, and depth.
res  Resolution of the faces.
alpha  Alpha channel for transparency.
lit  logical, specifying if rgl lighting calculation should take place.
shininess  Properties for rgl lighting calculation.
textype  Use "rgba" when sure texture will have alpha transparency. Use "rgb" when sure texture will not have alpha transparency (in particular rgl's WebGL export will likely work better). If NA we will read the texture and figure out a reasonable value.

Value

A numeric vector of rgl object IDs.
See Also

See rgl-package for more information about the rgl package. See rgl::material3d() for more info about setting rgl material properties. See geometry_utils for a discussion of the 3D rotation parameterization.

Examples

```r
if (requireNamespace("rgl", quietly = TRUE) && all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$piecepack
  piece3d("tile_back", suit = 3, rank = 3, cfg = cfg, x = 0, y = 0, z = 0)
  piece3d("coin_back", suit = 4, rank = 2, cfg = cfg, x = 0.5, y = 0.5, z = 0.25)
  piece3d("pawn_top", suit = 1, cfg = cfg, x = -0.5, y = 0.5, z = 0.6)
  piece3d("die_face", suit = 3, cfg = cfg, x = -0.5, y = -0.5, z = 0.375)
  piece3d("pyramid_top", suit = 2, rank = 3, cfg = cfg, x = 1.5, y = 0.0, z = 0.31875)
  invisible(NULL)
}
```

---

### piecepackr-defunct

#### Defunct functions

These functions are Defunct and have been removed from piecepackr.

#### Usage

- `halmaGrob(...)`
- `kiteGrob(...)`
- `pyramidGrob(...)`
- `convexGrobFn(...)`
- `concaveGrobFn(...)`
- `gridlinesGrob(...)`
- `matGrob(...)`
- `checkersGrob(...)`
- `hexlinesGrob(...)`
- `get_shape_grob_fn(...)`
piece_mesh

Arguments

... Ignored

Details

1. For get_shape_grob_fn use pp_shape()$shape instead.
2. For gridlinesGrob() use pp_shape()$gridlines() instead.
3. For matGrob() use pp_shape()$mat() instead.
4. For checkersGrob() use pp_shape()$checkers() instead.
5. For hexlinesGrob() use pp_shape()$hexlines() instead.
6. For halmaGrob() use pp_shape("halma")$shape() instead.
7. For kiteGrob() use pp_shape("kite")$shape() instead.
8. For pyramidGrob() use pp_shape("pyramid")$shape() instead.
9. For convexGrobFn(n, t) use pp_shape(paste0("convex", n), t)$shape instead.
10. For concaveGrobFn(n, t, r) use pp_shape(paste0("concave", n), t, r)$shape instead.

piece_mesh
Create rayvertex board game piece objects

Description

piece_mesh() creates 3d board game piece objects for use with the rayvertex package.

Usage

piece_mesh(
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = pp_cfg(),
  x = 0,
  y = 0,
  z = NA,
  angle = 0,
  axis_x = 0,
  axis_y = 0,
  width = NA,
  height = NA,
  depth = NA,
  envir = NULL,
...,
  scale = 1,
  res = 72
)
piece_mesh

Arguments

- **piece_side**: A string with piece and side separated by a underscore e.g. "coin_face"
- **suit**: Number of suit (starting from 1).
- **rank**: Number of rank (starting from 1)
- **cfg**: Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- **x**: Where to place piece on x axis of viewport
- **y**: Where to place piece on y axis of viewport
- **z**: z-coordinate of the piece. Has no effect if op_scale is 0.
- **angle**: Angle (on xy plane) to draw piece at
- **axis_x**: First coordinate of the axis unit vector.
- **axis_y**: Second coordinate of the axis unit vector.
- **width**: Width of piece
- **height**: Height of piece
- **depth**: Depth (thickness) of piece. Has no effect if op_scale is 0.
- **envir**: Environment (or named list) containing configuration list(s).
- **...**: Ignored.
- **scale**: Multiplicative scaling factor to apply to width, height, and depth.
- **res**: Resolution of the faces.

Value

A rayvertex object.

See Also

See [https://www.rayvertex.com](https://www.rayvertex.com) for more information about the rayvertex package. See [geometry_utils](https://CRAN.R-project.org/package=geometry_utils) for a discussion of the 3D rotation parameterization.

Examples

```r
# May take more than 5 seconds on CRAN servers
if (requireNamespace("rayvertex", quietly = TRUE) && all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$piecepack
  rs <- function(shape) {
    opt <- options(cores = getOption("Ncpus"))
    light <- rayvertex::directional_light(c(0, 0, 1))
    rayvertex::rasterize_scene(shape, light_info = light)
    options(opt)
  }
  rs(piece_mesh("tile_face", suit = 3, rank = 3, cfg = cfg))
}
if (requireNamespace("rayvertex", quietly = TRUE) && all(capabilities(c("cairo", "png")))) {
  rs(piece_mesh("coin_back", suit = 4, rank = 2, cfg = cfg))
}
```
pmap_piece

Create graphics using data frame input

Description

pmap_piece() operates on the rows of a data frame applying .f to each row (usually grid.piece).

Usage

pmap_piece(
  .l,  
  .f = pieceGrob,  
  ...,  
  cfg = getOption("piecepackr.cfg"),  
  envir = getOption("piecepackr.envir"),  
  trans = getOption("piecepackr.trans"),  
  draw = TRUE,  
  name = NULL,  
  gp = NULL,  
  vp = NULL
)

Arguments

.l  A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

.f  Function to be applied to .l after adjustments to cfg and envir and the application of trans. Usually grid.piece(), pieceGrob(), piece3d(), or piece().

...  Extra arguments to pass to .f.

cfg  Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base:::dynGet().

envir  Environment (or named list) containing configuration list(s).

trans  Function to modify .l before drawing. Default (NULL) is to not modify .l. op_transform can help with using an oblique projection (i.e. op_scale over 0).

draw  A logical value indicating whether graphics output should be produced.

name  A character identifier (for grid)

gp  An object of class “gpar”.

vp  A grid viewport object (or NULL).
Details

`pmap_piece()` differs from `purrr::pmap()` in a few ways:

1. If `cfg` and/or `envir` are missing attempts to set reasonable defaults.
2. If not NULL will first apply function `trans` to `.l`.
3. If the output of `.f` is a grid grob object then `pmap_piece` will return a `gTree` object with specified name, `gp`, and `vp` values and if `draw` is true draw it.
4. If `.l` lacks a name column or if name column is non-unique attempts to generate a reasonable new default name column and use that to name the return `gTree` children or list values.

See Also

`render_piece()` is a higher-level function that wraps this function.

Examples

```r
if (requireNamespace("grid", quietly = TRUE) & piecepackr:::device_supports_unicode()) {
  dark_colorscheme <- list(suit_color="darkred,black,darkgreen,darkblue,black", invert_colors.suited=TRUE, border_color="black", border_lex=2)
  traditional_ranks <- list(use_suit_as_ace=TRUE, rank_text="a,2,3,4,5")
  cfg3d <- list(width.pawn=0.75, height.pawn=0.75, depth.pawn=1, dm_text.pawn="", shape.pawn="convex6", invert_colors.pawn=TRUE, edge_color.coin="tan", edge_color.tile="tan")
  cfg <- pp_cfg(c(dark_colorscheme, traditional_ranks, cfg3d))
  grid::grid.newpage()
  df_tiles <- data.frame(piece_side="tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1), suit=NA, angle=NA, z=NA, stringsAsFactors=FALSE)
  df_coins <- data.frame(piece_side="coin_back", x=rep(4:1, 4), y=rep(4:1, each=4), suit=1:16%%2+rep(c(1,3), each=8), angle=rep(c(180,0), each=8), z=1/4+1/16, stringsAsFactors=FALSE)
  df <- rbind(df_tiles, df_coins)
  pmap_piece(df, cfg=cfg, op_scale=0.5, default.units="in")
}
```

**pp_cfg**

*Configuration list R6 object*

Description

`pp_cfg()` and `as_pp_cfg()` create piecepack configuration list R6 objects. `is_pp_cfg()` returns `TRUE` if object is a piecepack configuration list R6 object. `as_list()` will convert it into a list.

Usage

```r
pp_cfg(cfg = list())

is_pp_cfg(cfg)

as_pp_cfg(cfg = list())
```
Arguments
cfg List of configuration options

Details

**pp_cfg** R6 class objects serve the following purposes:

- Customize the appearance of pieces drawn by `grid.piece()`.
- Speed up the drawing of graphics through use of caching.
- Allow the setting and querying of information about the board game components that maybe of use to developers:
  - Number of suits
  - Number of ranks
  - Suit colors
  - Which types of components are included and/or properly supported
  - What would be a good color to use when adding annotations on top of these components.
  - Title, Description, Copyright, License, and Credit metadata

**pp_cfg R6 Class Method Arguments**

- `piece_side` A string with piece and side separated by a underscore e.g. "coin_face".
- `suit` Number of suit (starting from 1).
- `rank` Number of rank (starting from 1).
- `type` Which type of grob to return, either "normal", "picture", "raster", or "transformation".
- `scale"scale" factor
- `alpha"alpha" value

**pp_cfg R6 Class Methods**

- `get_grob()` Returns a `grid` "grob" for drawing the piece.
- `get_piece_opt()` Returns a list with info useful for drawing the piece.
- `get_suit_color()` Returns the suit colors.
- `get_width()`, `get_height()`, `get_depth()` Dimensions (of the bounding cube) of the piece in inches

**pp_cfg R6 Class Fields and Active Bindings**

- `annotation_color` Suggestion of a good color to annotate with
- `cache` Cache object which stores intermediate graphical calculations. Default is a memory-cache that does not prune. This can be replaced by another cache that implements the cache API used by the `cachem` package
- `cache_grob` Whether we should cache (2D) grobs
- `cache_grob_with_bleed_fn` Whether we should cache the grob with bleed functions
- `cache_piece_opt` Whether we should cache piece opt information
cache_op_fn  Whether we should cache the oblique projection functions

Whether we should cache any 3D rendering functions

copyright  Design copyright information
credit  Design credits
description  Design description

fontfamily  Main font family

has_bits  Whether we should assume this supports "bit" pieces
has_boards  Whether we should assume this supports "board" pieces
has_cards  Whether we should assume this supports "card" pieces
has_coins  Whether we should assume this supports "coin" pieces
has_dice  Whether we should assume this supports "die" pieces

has_matchsticks  Whether we should assume this supports "matchstick" pieces

has_pawns  Whether we should assume this supports "pawn" pieces

has_piecepack  Binding which simultaneously checks/sets has_coins, has_tiles, has_pawns, has_dice

has_pyramids  Whether we should assume this supports "pyramid" pieces

has_saucers  Whether we should assume this supports "saucer" pieces

has_tiles  Whether we should assume this supports "tile" pieces

spdx_id  SPDX Identifier for graphical design license. See https://spdx.org/licenses/ for full list.
title  Design title

Defunct pp_cfg R6 Class attributes which have been removed

  cache_shadow  Use cache_op_fn instead

  i_unsuit  Instead add 1L to n_suits

get_pictureGrob()  Use get_grob(..., type = "picture") instead

get_shadow_fn  get_op_grob() returns complete oblique projection grob

See Also

game_systems()  for functions that return configuration list objects for several game systems.

Examples

cfg <- pp_cfg(list(invert_colors=TRUE))
as.list(cfg)
is_pp_cfg(cfg)
as_pp_cfg(list(suit_color="darkred,black,darkgreen,darkblue,grey"))
cfg$get_suit_color(suit=3)
cfg$annotation_color
pp_shape

Shape object for generating various grobs

Description

pp_shape() creates an R6 object with methods for creating various shape based grobs.

Usage

pp_shape(label = "rect", theta = 90, radius = 0.2, back = FALSE)

Arguments

label Label of the shape. One of
  "circle" Circle.
  "convexN" An N-sided convex polygon. theta controls which direction the first vertex is drawn.
  "concaveN" A "star" (concave) polygon with N "points". theta controls which direction the first point is drawn. radius controls the distance of the "inner" vertices from the center.
  "halma" A 2D outline of a "Halma pawn".
  "kite" "Kite" quadrilateral shape.
  "meeple" A 2D outline of a "meeple".
  "oval" Oval.
  "pyramid" An "Isosceles" triangle whose base is the bottom of the viewport. Typically used to help draw the face of the "pyramid" piece.
  "rect" Rectangle.
  "roundrect" "Rounded" rectangle. radius controls curvature of corners.

theta convex and concave polygon shapes use this to determine where the first point is drawn.
radius concave polygon and roundrect use this to control appearance of the shape.
back Whether the shape should be reflected across a vertical line in the middle of the viewport.

Details

pp_shape objects serve the following purposes:

1. Make it easier for developers to customize game piece appearances either through a "grob_fn" or "op_grob_fn" styles in pp_cfg() or manipulate a piece post drawing via functions like grid::grid.edit().
2. Used internally to generate piecepackr’s built-in game piece grobs.

pp_shape R6 Class Method Arguments

mat_width Numeric vector of mat widths.
clip “clip grob” to perform polyclip operation with. See gridGeometry::grid.polyclip() for more info.
op Polyclip operation to perform. See gridGeometry::grid.polyclip() for more info.
pattern Pattern to fill in shape with. See gridpattern::patternGrob() for more info.
... Passed to gridpattern::patternGrob().
name Grid grob name value.
gp Grid gpar list. See grid::gpar() for more info.
vp Grid viewport or NULL.

pp_shape R6 Class Methods

checkers(name = NULL, gp = gpar(), vp = NULL) Returns a grob of checkers for that shape.
gridlines(name = NULL, gp = gpar(), vp = NULL) Returns a grob of gridlines for that shape.
hexlines(name = NULL, gp = gpar(), vp = NULL) Returns a grob of hexlines for that shape.
mat(mat_width = 0, name = NULL, gp = gpar(), vp = NULL) Returns a grob for a matting “mat” for that shape.
pattern(pattern = "stripe", ..., name = NULL, gp = gpar(), vp = NULL) Fills in the shape’s npc_coords with a pattern. See gridpattern::patternGrob() for more information.
polyclip(clip, op = "intersection", name = NULL, gp = gpar(), vp = NULL) Returns a grob that is an "intersection", "minus", "union", or "xor" of another grob. Note unlike gridGeometry::polyclipGrob it can directly work with a pieceGrob "clip grob" argument.
shape(name = NULL, gp = gpar(), vp = NULL) Returns a grob of the shape.

pp_shape R6 Class Active Bindings

label The shape’s label.
theta The shape’s theta.
radius The shape’s radius.
back A boolean of whether this is the shape’s “back” side.
npc_coords A named list of “npc” coordinates along the perimeter of the shape.
Examples

```r
if (require("grid", quietly = TRUE)) {
  gp <- gpar(col="black", fill="yellow")
  rect <- pp_shape(label="rect")
  convex6 <- pp_shape(label="convex6")
  circle <- pp_shape(label="circle")

  pushViewport(viewport(x=0.25, y=0.75, width=1/2, height=1/2))
  grid.draw(rect$shape(gp=gp))
  grid.draw(rect$gridlines(gp=gpar(col="blue", lex=4)))
  grid.draw(rect$hexlines(gp=gpar(col="green")))
  popViewport()

  pushViewport(viewport(x=0.75, y=0.75, width=1/2, height=1/2))
  grid.draw(convex6$shape(gp=gp))
  grid.draw(convex6$checkers(gp=gpar(fill="blue")))
  popViewport()

  pushViewport(viewport(x=0.25, y=0.25, width=1/2, height=1/2))
  grid.draw(circle$shape(gp=gp))
  grid.draw(circle$mat(mat_width=0.2, gp=gpar(fill="blue")))
  popViewport()

  pushViewport(viewport(x=0.75, y=0.25, width=1/2, height=1/2))
  grid.draw(rect$shape(gp=gp))
  grid.draw(rect$mat(mat_width=c(0.2, 0.1, 0.3, 0.4), gp=gpar(fill="blue")))
  popViewport()
}
if (require("grid", quietly = TRUE)) {
  grid.newpage()
  vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("halma")$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("pyramid")$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("kite")$shape(gp=gp, vp=vp))
  vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("meeple")$shape(gp=gp, vp=vp))
}
if (require("grid", quietly = TRUE)) {
  grid.newpage()
  vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("convex3", 0)$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("convex4", 90)$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("convex5", 180)$shape(gp=gp, vp=vp))
  vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("convex6", 270)$shape(gp=gp, vp=vp))
}
```
if (require("grid", quietly = TRUE)) {
  grid.newpage()
  vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("concave3", 0, 0.1)$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
  grid.draw(pp_shape("concave4", 90, 0.2)$shape(gp=gp, vp=vp))
  vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("concave5", 180, 0.3)$shape(gp=gp, vp=vp))
  vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
  grid.draw(pp_shape("concave6", 270)$shape(gp=gp, vp=vp))
}
if (require("grid", quietly = TRUE) && requireNamespace("gridpattern", quietly = TRUE)) {
  grid.newpage()
  hex <- pp_shape("convex6")
  gp <- gpar(fill = c("blue", "yellow", "red"), col = "black")
  grid.draw(hex$pattern("polygon_tiling", gp = gp, spacing = 0.1,
                       type = "truncated_trihexagonal"))
  gp <- gpar(fill = "black", col = NA)
  grid.draw(hex$mat(mat_width = 0.025, gp = gp))
}

### pp_utils

#### Miscellaneous piecepackr utility functions

---

**Description**

`cleave` converts a delimiter separated string into a vector. `inch(x)` is equivalent to `unit(x, "in")`. 

`is_color_invisible` tells whether the color is transparent (and hence need not be drawn).

**Usage**

```r
is_color_invisible(col)

inch(inches)

cleave(s, sep = ",", float = FALSE, color = FALSE)

file2grob(file, distort = TRUE)
```

**Arguments**

- `col` - Color
- `inches` - Number representing number of inches
- `s` - String to convert
- `sep` -Delimiter (defaults to ",")
- `float` - If TRUE cast to numeric
color  
if TRUE convert empty strings to "transparent"

file  
Filename of image

distort  
Logical value of whether one should preserve the aspect ratio or distort to fit the area it is drawn in

Examples

cleave("0.5,0.2,0.4,0.5", float=TRUE)
cleave("black,darkred,#050EAA,,", color=TRUE)

is_color_invisible("transparent")
is_color_invisible(NA)
is_color_invisible("blue")
is_color_invisible("#05AE9C")

if (requireNamespace("grid", quietly = TRUE)) {
  identical(inch(1), grid::unit(1, "inch"))
}

render_piece

render_piece() renders an image of game pieces to a file or graphics device. It is a wrapper around pmap_piece() that can auto-size files and graphic devices, apply axes offsets, annotate coordinates, and set up rayrender/rayvertex scenes.

Usage

render_piece(df, file = NULL, ...,
.f = piecepackr::grid.piece,
cfg = getOption("piecepackr.cfg", NULL),
envir = getOption("piecepackr.envir", game_systems("sans")),
width = NULL,
height = NULL,
ppi = 72,
bg = "white",
xoffset = NULL,
yoffset = NULL,
new_device = TRUE,
dev = NULL,
dev.args = list(res = ppi, bg = bg, units = "in"),
render_piece

annotate = FALSE,
annotation_scale = NULL
)

Arguments

df A data frame of game piece information with (at least) the named columns
    "piece_side", "x", and "y".
file Filename to save image unless NULL in which case it either uses the current
    graphics device or opens a new device (depending on new_device argument).
... Arguments to pmap_piece()
.f Low level graphics function to use e.g. grid.piece(), piece3d(), piece_mesh(),
    or piece().
cfg A piecepackr configuration list
envir Environment (or named list) of piecepackr configuration lists
width Width of image (in inches). Inferred by default.
height Height of image (in inches). Inferred by default.
ppi Resolution of image in pixels per inch.
bg Background color (use "transparent" for transparent)
xoffset Number to add to the x column in df. Inferred by default.
yoffset Number to add to the y column in df. Inferred by default.
new_device If file is NULL should we open up a new graphics device?
dev Graphics device function to use. If NULL infer a reasonable choice.
dev.args Additional arguments to pass to dev (besides filename, width, and height).
    Will filter out any names that aren't in formals(dev).
annotate If TRUE or "algebraic" annotate the plot with "algebraic" coordinates, if FALSE
    or "none" don't annotate, if "cartesian" annotate the plot with "cartesian" co-
    ordinates.
annotation_scale Multiplicative factor that scales (stretches) any annotation coordinates. By de-
    fault uses attr(df, "scale_factor") %||% 1.

Value

An invisible list of the dimensions of the image, as a side effect saves a graphic

See Also

This function is a wrapper around pmap_piece().
Examples

def_board <- data.frame(piece_side = "board_face", suit = 3, rank = 5,
                        x = 3.0, y = 3.0, stringsAsFactors = FALSE)
def_w <- data.frame(piece_side = "bit_face", suit = 6, rank = 1,
                     x = rep(1:5, 2), y = rep(1:2, each=5),
                     stringsAsFactors = FALSE)
def_b <- data.frame(piece_side = "bit_face", suit = 1, rank = 1,
                     x = rep(1:5, 2), y = rep(4:5, each=5),
                     stringsAsFactors = FALSE)
df <- rbind(df_board, df_w, df_b)
df$cfg <- "checkers1"

if (requireNamespace("grid", quietly = TRUE)) {
    render_piece(df, new_device = FALSE)
} else {
    grid::grid.newpage()
    render_piece(df, new_device = FALSE,
                 op_scale = 0.5, trans = op_transform,
                 annotate = "algrebraic")
}

## Not run: # May take more than 5 seconds on CRAN servers
if (require(rayvertex)) {
    envir3d <- game_systems("sans3d")
    render_piece(df, .f = piece_mesh, envir = envir3d,
                 op_scale = 0.5, trans = op_transform)
}

## End(Not run)

save_ellipsoid_obj

Alternative Wavefront OBJ file generators

Description

These are alternative Wavefront OBJ generators intended to be used as a `obj_fn` attribute in a `pp_cfg()` "configuration list". `save_ellipsoid_obj` saves an ellipsoid with a color equal to that piece’s background_color. `save_peg_doll_obj` saves a "peg doll" style doll with a color equal to that piece’s edge_color with a "pawn belt" around it’s waste from that suit’s and rank’s belt_face.

Usage

```
save_ellipsoid_obj(
    piece_side = "bit_face",
    suit = 1,
    rank = 1,
    cfg = getOption("piecepackr.cfg", pp_cfg()),
    ...
    x = 0,
```
save_ellipsoid_obj

```r
y = 0,
z = 0,
angle = 0,
axis_x = 0,
axis_y = 0,
width = NA,
height = NA,
deepth = NA,
filename = tempfile(fileext = ".obj"),
subdivide = 3
```

```r
save_peg_doll_obj(
    piece_side = "pawn_top",
suit = 1,
rank = 1,
cfg = getOption("piecepackr.cfg", pp_cfg()),
...
    x = 0,
y = 0,
z = 0,
angle = 0,
axis_x = 0,
axis_y = 0,
width = NA,
height = NA,
deepth = NA,
filename = tempfile(fileext = ".obj"),
res = 72
)
```

**Arguments**

- `piece_side` A string with piece and side separated by a underscore e.g. "coin_face"
- `suit` Number of suit (starting from 1).
- `rank` Number of rank (starting from 1)
- `cfg` Piecepack configuration list or `pp_cfg` object, a list of `pp_cfg` objects, or a character vector referring to names in `envir` or a character vector referring to object names that can be retrieved by `base::dynGet()`.
- `...` Ignored.
- `x` Where to place piece on x axis of viewport
- `y` Where to place piece on y axis of viewport
- `z` z-coordinate of the piece. Has no effect if `op_scale` is 0.
- `angle` Angle (on xy plane) to draw piece at
- `axis_x` First coordinate of the axis unit vector.
- `axis_y` Second coordinate of the axis unit vector.
save_piece_images

Description

Saves images of all individual piecepack pieces.

Usage

save_piece_images(
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  directory = tempdir(),
  format = "svg",
  angle = 0
)

Arguments

  cfg  Piecepack configuration list
  directory Directory where to place images
  format   Character vector of formats to save images in
  angle   Numeric vector of angles to rotate images (in degrees)

Examples

# May take more than 5 seconds on CRAN server
if (all(capabilities(c("cairo", "png")))) {
  cfg <- pp_cfg(list(suit_color="darkred,black,darkgreen,darkblue, grey"))
  save_piece_images(cfg, directory=tempdir(), format="svg", angle=0)
  save_piece_images(cfg, directory=tempdir(), format="png", angle=90)
}
save_piece_obj

Save Wavefront OBJ files of board game pieces

Description

save_piece_obj saves Wavefront OBJ files (including associated MTL and texture image) of board game pieces.

Usage

save_piece_obj(
  piece_side = "tile_face",
  suit = 1,
  rank = 1,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  ...
  x = 0,
  y = 0,
  z = 0,
  angle = 0,
  axis_x = 0,
  axis_y = 0,
  width = NA,
  height = NA,
  depth = NA,
  filename = tempfile(fileext = ".obj"),
  scale = 1,
  res = 72
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>piece_side</td>
<td>A string with piece and side separated by a underscore e.g. &quot;coin_face&quot;</td>
</tr>
<tr>
<td>suit</td>
<td>Number of suit (starting from 1).</td>
</tr>
<tr>
<td>rank</td>
<td>Number of rank (starting from 1)</td>
</tr>
<tr>
<td>cfg</td>
<td>Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet()</td>
</tr>
<tr>
<td></td>
<td>(Continued)</td>
</tr>
<tr>
<td></td>
<td>Ignored.</td>
</tr>
<tr>
<td>x</td>
<td>Where to place piece on x axis of viewport</td>
</tr>
<tr>
<td>y</td>
<td>Where to place piece on y axis of viewport</td>
</tr>
<tr>
<td>z</td>
<td>z-coordinate of the piece. Has no effect if op_scale is 0.</td>
</tr>
<tr>
<td>angle</td>
<td>Angle (on xy plane) to draw piece at</td>
</tr>
<tr>
<td>axis_x</td>
<td>First coordinate of the axis unit vector.</td>
</tr>
</tbody>
</table>
save_print_and_play

axis_y  Second coordinate of the axis unit vector.
width   Width of piece
height  Height of piece
depth   Depth (thickness) of piece. Has no effect if op_scale is 0.
filename Name of Wavefront OBJ object.
scale   Multiplicative scaling factor to apply to width, height, and depth.
res     Resolution of the faces.

Value
A list with named elements "obj", "mtl", "png" with the created filenames.

See Also
See geometry_utils for a discussion of the 3D rotation parameterization.

Examples
if (all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$dominoes
  files <- save_piece_obj("tile_face", suit = 3+1, rank=6+1, cfg = cfg)
  print(files)
}

save_print_and_play  Save piecepack print-and-play (PnP) file

Description
Save piecepack print-and-play (PnP) file

Usage
save_print_and_play(
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  output_filename = "piecepack.pdf",
  size = c("letter", "A4", "A5", "4x6"),
  pieces = NULL,
  arrangement = c("single-sided", "double-sided"),
  dev = NULL,
  dev.args = list(family = cfg$fontfamily, onefile = TRUE, units = "in", bg = "white",
                  res = 300),
  quietly = FALSE,
  ...
  bleed = FALSE,
  size_bleed = NULL
)
Arguments

`cfg` Piecepack configuration list or `pp_cfg` object

`output_filename` Filename for print-and-play file

`size` PnP output size (currently supports either "letter", "A4", "A5", or "4x6"). This is the targeted "trim" size of the print-and-play file (`size_bleed` can be used to make the print-and-play file larger than this). Size "4x6" currently only supports `pieces = "piecepack"` and doesn't support `bleed = TRUE`. "A5" is in "portrait" mode whereas the other sizes are in "landscape" mode.

`pieces` Character vector of desired PnP pieces. Supports "piecepack", "matchsticks", "pyramids", "subpack", or "all". If NULL and combination of `size` / `bleed` values supports "matchsticks" and "pyramids" then defaults to c("piecepack", "pyramids", "matchsticks") else just "piecepack".

`arrangement` Either "single-sided" or "double-sided". Ignored if `size = "4x6"`.

`dev` Graphics device function to use. If `NULL` infer a reasonable choice.

`dev.args` Additional arguments to pass to `dev` (besides `filename`, `width`, and `height`). Will filter out any names that aren't in `formals(dev)`.

`quietly` Whether to hide messages about missing metadata in the provided configuration.

`...` Currently ignored.

`bleed` If `TRUE` produce a variant print-and-play file with "bleed" zones and "crop marks" around game pieces. Currently only supports `pieces = "piecepack"` and doesn't support `size = "4x6"`.

`size_bleed` A list with names "top", "right", "bottom", "left" containing numeric values indicating the inches "bleed" to add to the size of the print-and-play layout. The default NULL means no such bleed added to "letter", "A4", "A5" layouts and a small bleed added to "4x6" layouts (1/16" to top/bottom and 3/32" to left/right). NB. multiply millimeters by 0.0393700787 to convert to inches. We currently don't support an asymmetric left/right bleed combined with `arrangement = "double-sided"`.

Examples

```r
# May take more than 5 seconds on CRAN servers
if (capabilities("cairo")) {
  cfg <- pp_cfg(list(invert_colors.suited=TRUE))
  cfg$description <- 'Piecepack with an "inverted" color scheme.'
  cfg$title <- 'Inverted piecepack'
  cfg$copyright <- "\u00a9 2022 Trevor L Davis. Some Right Reserved."
  cfg$spdx_id <- "CC-BY-4.0"
  cfg$credit <- ""

  file <- tempfile("my_pnp_file", fileext = ".pdf")
  file_ds <- tempfile("my_pnp_file_ds", fileext = ".pdf")
  file_a4 <- tempfile("my_pnp_file_a4", fileext = ".pdf")
  file_a5 <- tempfile("my_pnp_file_a5", fileext = ".pdf")
```

scale_x_piece

save_print_and_play(cfg, file)
save_print_and_play(cfg, file_ds, arrangement="double-sided")
save_print_and_play(cfg, file_a4, size="A4", pieces="all")
save_print_and_play(cfg, file_a5, size="A5")
}

scale_x_piece
ggplot2 game diagram scales

Description

scale_x_piece() and scale_y_piece() are wrappers around ggplot2::scale_x_continuous() and ggplot2::scale_y_continuous() with "better" defaults for board game diagrams. label_letter() labels breaks with letters and label_counting() labels breaks with positive integers to more easily generate (i.e. chess) algebraic notation coordinates. breaks_counting() generates breaks of just the positive integers within the limits.

Usage

scale_x_piece(
    ..., 
    name = NULL, 
    breaks = breaks_counting(), 
    minor_breaks = NULL, 
    labels = label_letter()
)

scale_y_piece(
    ..., 
    name = NULL, 
    breaks = breaks_counting(), 
    minor_breaks = NULL, 
    labels = label_counting()
)

label_letter()

label_counting()

breaks_counting()

Arguments

... Passed to ggplot2::scale_x_continuous() or ggplot2::scale_y_continuous().

name The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
breaks

One of:
- NULL for no breaks
- waiver() for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang `lambda` function notation.

minor_breaks

One of:
- NULL for no minor breaks
- waiver() for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang `lambda` function notation.

labels

One of:
- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.

Value

`scales::scale_x_piece()` and `scales::scale_y_piece()` return `ggplot2` scale objects. `scales::label_letter()` and `scales::label_counting()` return functions suitable for use with the `labels` scale argument. `scales::breaks_counting()` returns a function suitable for use with the `breaks` scale argument.

Examples

```r
if (require("ggplot2", quietly = TRUE) && require("tibble", quietly = TRUE)) {
  envir <- game_systems("sans")
  df_board <- tibble(piece_side = "board_face", suit = 3, rank = 8,
                      x = 4.5, y = 4.5)
  df_w <- tibble(piece_side = "bit_face", suit = 6, rank = 1,
                  x = rep(1:8, 2), y = rep(1:2, each=8))
  df_b <- tibble(piece_side = "bit_face", suit = 1, rank = 1,
                  x = rep(1:8, 2), y = rep(7:8, each=8))
  df <- rbind(df_board, df_w, df_b)

  # `cfg` must be a character vector for `geom_piece`
  ggplot(df, aes_piece(df)) +
  geom_piece(cfg = "checkers!", envir = envir) +
  coord_fixed() +
  scale_x_piece() +
  scale_y_piece() +
```
spdx_license_list

```r
theme_minimal(28) +
theme(panel.grid = element_blank())
```

---

**spdx_license_list**  
SPDX License List data

---

**Description**

`spdx_license_list` is a data frame of SPDX License List data.

**Usage**

`spdx_license_list`

**Format**

A data frame with eight variables:

- **id**  
  SPDX Identifier.

- **name**  
  Full name of license. For Creative Commons licenses these have been tweaked from the SPDX version to more closely match the full name used by Creative Commons Foundation.

- **url**  
  URL for copy of license located at spdx.org.

- **fsf**  
  Is this license considered Free/Libre by the FSF?

- **osi**  
  Is this license OSI approved?

- **deprecated**  
  Has this SPDX Identifier been deprecated by SPDX?

- **badge**  
  Filename of appropriate “button mark” badge (if any) located in `system.file("extdata/badges", package = "piecepackr")`.

- **url_alt**  
  Alternative URL for license. Manually created for a subset of Creative Commons licenses. Others taken from [https://github.com/sindresorhus/spdx-license-list](https://github.com/sindresorhus/spdx-license-list).

**See Also**

See [https://spdx.org/licenses/](https://spdx.org/licenses/) for more information.
Index

* datasets
  spdx_license_list, 49

AA_to_R, 5
aabb_piece, 4
aes(), 15
aes_piece (geom_piece), 15
animate_piece, 7
as_pp_cfg (pp_cfg), 32
base::options(), 3
basicPieceGrob (basicPieceGrobs), 9
basicPieceGrobs, 9
borders(), 16
breaks_counting (scale_x_piece), 47
checkersGrob (piecepackr-defunct), 28
cleave (pp_utils), 38
concaveGrobFn (piecepackr-defunct), 28
convexGrobFn (piecepackr-defunct), 28
cropmarkGrob (grid.cropmark), 17
ellipse3d, 43
file2grob (pp_utils), 38
font_utils, 10
fortify(), 15
game_systems, 11
game_systems(), 34
geom_piece, 15
geometry_utils, 26, 28, 30, 45
gometry_utils (AA_to_R), 5
get_embedded_font (font_utils), 10
get_shape_grob_fn (piecepackr-defunct), 28
ggplot(), 15
ggplot2::scale_x_continuous(), 17, 47
ggplot2::scale_y_continuous(), 17, 47
ggrid.cropmark, 17
ggrid.piece, 20
ggrid.piece(), 8, 31, 40
ggrid::gpar(), 36
ggrid::unit(), 19, 22
gridGeometry::grid.polygon(), 36
gridlinesGrob (piecepackr-defunct), 28
gridpattern::patternGrob(), 36
halmaGrob (piecepackr-defunct), 28
has_font (font_utils), 10
hexlinesGrob (piecepackr-defunct), 28
inch (pp_utils), 38
is_color_invisible (pp_utils), 38
is_pp_cfg (pp_cfg), 32
kiteGrob (piecepackr-defunct), 28
label_counting (scale_x_piece), 47
label_letter (scale_x_piece), 47
lambda, 48
matGrob (piecepackr-defunct), 28
op_transform, 23
picturePieceGrobFn (basicPieceGrobs), 9
piece, 24
piece(), 8, 31, 40, 43
piece3d, 26
piece3d(), 8, 31, 40, 43
piece_mesh, 29
piece_mesh(), 8, 40, 43
pieceGrob (grid.piece), 20
pieceGrob(), 16, 17, 31
piecepackr (piecpackr-package), 3
piecepackr-defunct, 28
piecepackr-package, 3
pmap_piece, 31
pmap_piece(), 22, 40
pp_cfg, 14, 32
pp_cfg(), 43
pp_shape, 35
pp_utils, 38
previewLayoutGrob(basicPieceGrobs), 9
pyramidGrob(piecepackr-defunct), 28
pyramidTopGrob(basicPieceGrobs), 9

R_to_AA(AA_to_R), 5
R_x(AA_to_R), 5
R_y(AA_to_R), 5
R_z(AA_to_R), 5
render_piece, 39
render_piece(), 32
rgl::material3d(), 28

save_ellipsoid_obj, 41
save_peg_doll_obj(save_ellipsoid_obj), 41
save_piece_images, 43
save_piece_obj, 44
save_piece_obj(), 43
save_print_and_play, 45
scale_x_piece, 47
scale_x_piece(), 17
scale_y_piece(scale_x_piece), 47
scale_y_piece(), 17
scales::extended_breaks(), 48
spdx_license_list, 49

to_degrees(AA_to_R), 5
to_hexpack(game_systems), 11
to_r(AA_to_R), 5
to_radians(AA_to_R), 5
to_subpack(game_systems), 11
to_t(AA_to_R), 5
to_x(AA_to_R), 5
to_y(AA_to_R), 5
transformation object, 48
Trig, 6