Package 'piecepackr'

August 10, 2020

Encoding UTF-8

Type Package

Title Board Game Graphics

Version 1.5.1

Description Functions to make board game graphics. By default makes game diagrams, animations, and ``Print & Play'' layouts for the 'piecepack' <http://www.ludism.org/ppwiki> but can be configured to make graphics for other board game systems.

License CC BY-SA 4.0

URL https://trevorldavis.com/piecepackr/,
https://github.com/piecepackr/piecepackr,
https://groups.google.com/forum/#!forum/piecepackr

BugReports https://github.com/piecepackr/piecepackr/issues

LazyLoad yes

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

Suggests magick, rayrender (>= 0.5.8), rgl (>= 0.100.46), testthat, vdiffr

SystemRequirements ghostscript

RoxygenNote 7.1.1

NeedsCompilation no

Author Trevor L Davis [aut, cre], Delapouite <https://delapouite.com/> [dtc] (Meeple shape extracted from ``Meeple icon'' <https://game-icons.net/1x1/delapouite/meeple.html> / ``CC BY 3.0'' <http://creativecommons.org/licenses/by/3.0/>)

Maintainer Trevor L Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2020-08-10 17:20:11 UTC

R topics documented:

`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

```
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

https://en.wikipedia.org/wiki/Axis-angle_representation for more details about the Axis-angle representation of 3D rotations. See Trig for R's built-in trigonometric functions.

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))
```

---

basicPieceGrobs *Piece Grob Functions*

Description

basicPieceGrob, pyramidTopGrob, and previewLayoutGrob are the default "grob" functions 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

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.
directory	Directory that picturePieceGrobFn will look in for piece graphics.
filename_fn	Function that takes arguments directory, piece_side, suit, and rank and returns the (full path) filename of the image that the function returned by picturePieceGrobFn should import.

Examples

```
if (require("grid")) {
   cfg <- pp_cfg(list(invert_colors=TRUE))

   pushViewport(viewport(width=unit(2, "in"), height=unit(2, "in")))
   grid.draw(basicPieceGrob("tile_face", suit=1, rank=3))
   popViewport()

   grid.newpage()
   pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
   grid.draw(basicPieceGrob("coin_back", suit=2, rank=0, cfg=cfg))
   popViewport()

   grid.newpage()
   pushViewport(viewport(width=unit(6, "in"), height=unit(6, "in")))
   grid.draw(previewLayoutGrob("preview_layout", suit=5, rank=0, cfg=cfg))
   popViewport()

   grid.newpage()
   pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
   grid.draw(pyramidTopGrob("pyramid_top", suit=3, rank=5))
   popViewport()


       directory <- tempdir()
       save_piece_images(cfg, directory=directory, format="svg", angle=0)
       cfg2 <- pp_cfg(list(grob_fn=picturePieceGrobFn(directory)))

       grid.newpage()
       pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
       grid.draw(pyramidTopGrob("pyramid_top", suit=3, rank=5, cfg=cfg2))
       popViewport()

}
```

`game_systems` *Standard game systems*

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)
```

```
to_hexpack(cfg = pp_cfg())

to_subpack(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).
cfg	List of configuration options

Details

Contains the following game systems:

checkers1, checkers2 Checkers and checkered boards in six color schemes. Checkers are represented 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.

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

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".

dual_piecepacks_expansion A companion piecepack with a special suit scheme. See https://trevorldavis.com/piecepackr/dual-piecepacks-pnp.html.

hexpack A hexagonal extrapolation of the piecepack designed by Nathan Morse and Daniel Wilcox. See https://boardgamegeek.com/boardgameexpansion/35424/hexpack.

meeples Standard 16mm x 16mm x 10mm "meeples" in six colors represented by a "bit".

piecepack A public domain game system invented by James "Kyle" Droscha. See http://www.ludism.org/ppwiki. Configuration also contains the following piecepack accessories:

- **piecepack dice cards** An accessory proposed by John Braley. See http://www.ludism.org/ppwiki/PiecepackDiceCards.
- **piecepack matchsticks** A public domain accessory developed by Dan Burkey. See http://www.ludism.org/ppwiki/PiecepackMatchsticks.
- **piecepack pyramids** A public domain accessory developed by Tim Schutz. See http://www.ludism.org/ppwiki/PiecepackPyramids.
- **piecepack saucers** A public domain accessory developed by Karol M. Boyle at Mesomorph Games. See https://web.archive.org/web/20190719155827/http://www.piecepack.org/Accessories.html.

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 fifth "suit" of 22 trump cards (1-21 plus an "excuse").

playing_cards_expansion A piecepack with the standard "French" playing card suits. See http://www.ludism.org/ppwiki/PlayingCardsExpansion.

subpack A mini piecepack. Designed to be used with the piecepack to make piecepack "stackpack" diagrams. See http://www.ludism.org/ppwiki/StackPack.

See Also

pp_cfg for information about the pp_cfg objects returned by game_systems.

Examples

```
  cfgs <- game_systems()
  names(cfgs)

if (require("grid")) {
  # standard dice
  grid.newpage()
  grid.piece("die_face", x=1:6, default.units="in", rank=1:6, suit=1:6,
             op_scale=0.5, cfg=cfgs$dice)

  # dominoes
  grid.newpage()
  colors <- c("black", "red", "green", "blue", "yellow", "white")
  cfg <- paste0("dominoes_", rep(colors, 2))
 grid.piece("tile_face", x=rep(4:1, 3), y=rep(2*3:1, each=4), suit=1:12, rank=1:12+1,
             cfg=cfg, default.units="in", envir=cfgs, op_scale=0.5)

  # various piecepack expansions
  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=c(1,2,1,2,2,1,2,1,4,3,4,3,3,4,3,4),
                    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$piecepack, op_scale=0.5, default.units="in")

  grid.newpage()
  df_coins <- data.frame(piece_side="coin_back", x=rep(4:1, 4), y=rep(4:1, each=4),
                        suit=c(1,4,1,4,4,1,4,1,2,3,2,3,3,2,3,2),
                    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")

  grid.newpage()
 pmap_piece(df, cfg = cfgs$dual_piecepacks_expansion, op_scale=0.5, default.units="in")
}
```

grid.piece *Draw board game pieces using grid*

Description

grid.piece draws board game pieces onto the graphics device. pieceGrob is its grid grob counterpart.

Usage

```
pieceGrob(
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = pp_cfg(),
  x = unit(0.5, "npc"),
  y = unit(0.5, "npc"),
  z = NA,
  angle = 0,
  use_pictureGrob = FALSE,
  width = NA,
  height = NA,
  depth = NA,
  op_scale = 0,
  op_angle = 45,
  default.units = "npc",
  envir = NULL,
  name = NULL,
  gp = NULL,
  vp = NULL,
  ...,
  scale = 1,
  alpha = 1
)

grid.piece(
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = pp_cfg(),
  x = unit(0.5, "npc"),
  y = unit(0.5, "npc"),
  z = NA,
  angle = 0,
  use_pictureGrob = FALSE,
  width = NA,
  height = NA,
```

```
  depth = NA,
  op_scale = 0,
  op_angle = 45,
  default.units = "npc",
  envir = NULL,
  name = NULL,
  gp = NULL,
  draw = TRUE,
  vp = NULL,
  ...,
  scale = 1,
  alpha = 1
)
```

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 of pp_cfg objects
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
use_pictureGrob	If TRUE instead of directly returning the grob first export to (temporary) svg and then re-import as a grImport2::pictureGrob. This is useful if drawing pieces really big or small and don't want to play with re-configuring fontsizes.
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).

...	Ignored.
scale	Multiplicative scaling factor to apply to width, height, and depth.
alpha	Alpha channel for transparency.
draw	A logical value indicating whether graphics output should be produced.

Value

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

See Also

pmap_piece which applies pieceGrob over rows of a data frame.

Examples

```
if (require("grid")) {
    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
    draw_pp_diagram(cfg=pp_cfg())

    # custom configuration, orthogonal projection
    grid.newpage()
    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")
    cfg <- c(dark_colorscheme, traditional_ranks)
    draw_pp_diagram(cfg=pp_cfg(cfg))

    # custom configuration, oblique projection
    grid.newpage()
    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(cfg, cfg3d))
    draw_pp_diagram(cfg=pp_cfg(cfg), op_scale=0.5)
}
```

`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 = pp_cfg(),
  envir = NULL,
  op_angle = 45,
  pt_thickness = 0.01,
  as_top = character(0)
)
```

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")`.

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

https://trevorldavis.com/piecepackr/3d-projections.html for more details and examples of oblique projections in `piecepackr`.

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))
pmap_piece(df, op_angle=135, trans=op_transform,
           op_scale=0.5, default.units="in")
```

---

piece *Create rayrender objects*

Description

piece creates 3d board game piece objects for use with the rayrender package.

Usage

```
piece(
  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
)
```

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 of pp_cfg objects
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

```
if (require("rayrender")) {
    cfg <- game_systems("sans3d")$piecepack
    render_scene(piece("tile_face", suit = 3, rank = 3, cfg = cfg))
    render_scene(piece("coin_back", suit = 4, rank = 2, cfg = cfg))
    render_scene(piece("pawn_face", suit = 2, cfg = cfg))
}
```

---

piece3d *Draw board game pieces using rgl*

Description

piece3d draws board games pieces using the rgl package.

Usage

```
piece3d(
  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,
  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 of pp_cfg objects
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.material for more info about setting rgl material properties. See geometry_utils for a discussion of the 3D rotation parameterization.

Examples

```
if ((Sys.getenv("TRAVIS") == "") && require("rgl")) {
    open3d()
    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.)
}
```

---

piecepackr-deprecated *Deprecated functions*

Description

These functions are Deprecated in this release of piecepackr, they will be marked as Defunct and removed in a future version.

Usage

```
halmaGrob(name = NULL, gp = gpar(), vp = NULL)

kiteGrob(name = NULL, gp = gpar(), vp = NULL)

pyramidGrob(name = NULL, gp = gpar(), vp = NULL)
```

```
convexGrobFn(n_vertices, t)

concaveGrobFn(n_vertices, t, r = 0.2)

gridlinesGrob(col, shape = "rect", shape_t = 90, lex = 1, name = NULL)

matGrob(col, shape = "rect", shape_t = 90, mat_width = 0, name = NULL)

checkersGrob(col, shape = "rect", shape_t = 90, name = NULL)

hexlinesGrob(col, shape = "rect", name = NULL)

get_shape_grob_fn(shape, shape_t = 90, shape_r = 0.2, back = FALSE)
```

Arguments

name	A character identifier (for grid)
gp	An object of class 'gpar'
vp	A grid viewport object (or NULL).
n_vertices	Number of vertices
t	Angle (in degrees) of first vertex of shape
r	Radial distance (from 0 to 0.5)
col	Color
shape	Label of shape
shape_t	Angle (in degrees) of first vertex of shape (ignored by many shapes).
lex	Scales width of line.
mat_width	Numeric vector of mat widths
shape_r	Radial distance (from 0 to 0.5) (ignored by most shapes)
back	Logical of whether back of the piece, in which case will reflect shape along vertical axis.

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.

Examples

```
if (require("grid")) {
    suppressWarnings({
        gp <- gpar(col="black", fill="yellow")
        pushViewport(viewport(x=0.25, y=0.75, width=1/2, height=1/2))
        grid.draw(get_shape_grob_fn("rect")(gp=gp))
        grid.draw(gridlinesGrob("blue", lex=4))
        grid.draw(hexlinesGrob("green"))
        popViewport()

        pushViewport(viewport(x=0.75, y=0.75, width=1/2, height=1/2))
        grid.draw(get_shape_grob_fn("convex6")(gp=gp))
        grid.draw(checkersGrob("blue", shape="convex6"))
        popViewport()

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

        pushViewport(viewport(x=0.75, y=0.25, width=1/2, height=1/2))
        grid.draw(get_shape_grob_fn("rect")(gp=gp))
        grid.draw(matGrob("blue", shape="rect", mat_width=c(0.2, 0.1, 0.3, 0.4)))
        popViewport()

        grid.newpage()
        gp <- gpar(col="black", fill="yellow")

        vp <- viewport(x=1/3-1/6, width=1/3)
        grid.draw(halmaGrob(gp=gp, vp=vp))
        vp <- viewport(x=2/3-1/6, width=1/3)
        grid.draw(pyramidGrob(gp=gp, vp=vp))
        vp <- viewport(x=3/3-1/6, width=1/3)
        grid.draw(kiteGrob(gp=gp, vp=vp))

        grid.newpage()
        vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
        grid.draw(convexGrobFn(3, 0)(gp=gp, vp=vp))
        vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
        grid.draw(convexGrobFn(4, 90)(gp=gp, vp=vp))
        vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
        grid.draw(convexGrobFn(5, 180)(gp=gp, vp=vp))
        vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
        grid.draw(convexGrobFn(6, 270)(gp=gp, vp=vp))

        grid.newpage()
        vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
        grid.draw(concaveGrobFn(3, 0, 0.1)(gp=gp, vp=vp))
        vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
        grid.draw(concaveGrobFn(4, 90, 0.2)(gp=gp, vp=vp))
        vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
        grid.draw(concaveGrobFn(5, 180, 0.3)(gp=gp, vp=vp))
```

```
        vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
        grid.draw(concaveGrobFn(6, 270)(gp=gp, vp=vp))
    }, classes = "deprecatedWarning")
}
```

`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 = NULL,
  envir = NULL,
  trans = NULL,
  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 `grid.piece_wrapper` 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`.
`...`	Extra arguments to pass to `.f`.
`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).
`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.

Examples

```
if (require("grid")) {
     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.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 creates piecepack configuration list R6 object. is_pp_cfg returns TRUE if object is a piecepack configuration list R6 object. as.list will convert it into a list.

Usage

```
pp_cfg(cfg = list())

is_pp_cfg(cfg)

as_pp_cfg(cfg = list())
```

Arguments

cfg	List of configuration options

Details

pp_cfg objects serve the following purposes:

1. Customize the appearance of pieces drawn by grid.piece.
2. Speed up the drawing of graphics through use of caching.
3. Allow the setting and querying of information about the board game components that maybe of use to developers
   (a) Number of suits
   (b) Number of ranks
   (c) Suit colors
   (d) Which types of components are included and/or properly supported
   (e) What would be a good color to use when adding annotations on top of these components.
   (f) Title, Description, Copyright, 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", or "raster".

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

See Also

https://trevorldavis.com/piecepackr/configuration-lists.html for more details about piecepackr configuration lists. 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
cfg$has_matchsticks
cfg$has_matchsticks <- TRUE
cfg$has_matchsticks
cfg$get_width("tile_back")
```

```
cfg$get_height("die_face")
cfg$get_depth("coin_face")

  cfg <- list()
  system.time(replicate(100, grid.piece("tile_face", 4, 4, cfg)))
  cfg <- pp_cfg(list())
  system.time(replicate(100, grid.piece("tile_face", 4, 4, cfg)))
```

---

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 grid.polyclip for more info.

op Polyclip operation to perform. See grid.polyclip for more info.

name Grid grob name value.

gp Grid gpar list. See 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.

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

```
if (require("grid")) {
    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()

    grid.newpage()
    gp <- gpar(col="black", fill="yellow")

    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))

    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))

    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))
}
```

pp_utils piecepackr *utility functions*

Description

get_embedded_font returns which font is actually embedded by cairo_pdf. 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

```
is_color_invisible(col)

get_embedded_font(font, char)

inch(inches)

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

file2grob(file, distort = TRUE)
```

Arguments

col	Color
font	A character vector of font(s) passed to the fontfamily argument of grid::gpar.
char	A character vector of character(s) to be embedded by grid::grid.text
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

Details

get_embedded_font depends on pdffonts being on the system path (on many OSes found in a poppler-utils package).

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

```
to_x(90, 1)
to_y(180, 0.5)
to_t(0, -1)
to_r(0.5, 0)

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

if (require("grid")) {
    grid.rect(width=inch(1), height=inch(3), gp=gpar(fill="blue"))
}
if ((Sys.which("pdffonts") != "") && capabilities("cairo")) {
    chars <- c("a", "\u2666")
    fonts <- c("sans", "Sans Noto", "Noto Sans", "Noto Sans Symbols2")
    get_embedded_font(fonts, chars)
}

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

---

save_piece_images *Save piecepack images*

Description

Saves images of all individual piecepack pieces.

Usage

```
save_piece_images(
  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

```
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 = 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

`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 of `pp_cfg` objects

...	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.
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

```
cfg <- game_systems("dejavu3d")$piecepack
files <- save_piece_obj("tile_face", suit = 3, rank = 3, 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 = pp_cfg(),
  output_filename = "piecepack.pdf",
  size = "letter",
  pieces = c("piecepack", "matchsticks", "pyramids"),
  arrangement = "single-sided"
)
```

Arguments

cfg	Piecepack configuration list
output_filename	Filename for print-and-play file
size	PnP output size (currently either "letter", "A4", or "A5")
pieces	Character vector of desired PnP pieces. Supports "piecepack", "matchsticks", "pyramids", "subpack", or "all".
arrangement	Either "single-sided" or "double-sided".

Examples

```
if (capabilities("cairo")) {
    cfg <- pp_cfg(list(invert_colors.suited=TRUE))
    save_print_and_play(cfg, "my_pnp_file.pdf")
    save_print_and_play(cfg, "my_pnp_file_ds.pdf", arrangement="double-sided")
    save_print_and_play(cfg, "my_pnp_file_A4.pdf", size="A4", pieces="all")
    save_print_and_play(cfg, "my_pnp_file_A5.pdf", size="A5")
    unlink("my_pnp_file.pdf")
    unlink("my_pnp_file_ds.pdf")
    unlink("my_pnp_file_A4.pdf")
    unlink("my_pnp_file_A5.pdf")
}
```

Index