Package: craft (via r-universe)

June 1, 2026

Title Extra Functions to Control Minecraft with RaspberryJuice API

Version 0.3.6

Date 2025-05-20

Description Extra functions to illustrate the use of the miner package to control Minecraft with the RaspberryJuice API.

Depends R (>= 3.5.0), miner

Imports Rmaze, igraph, imager, stats

Suggests testthat, knitr, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/kbroman/craft

BugReports https://github.com/kbroman/craft/issues

VignetteBuilder knitr

Roxygen list(markdown=TRUE)

Remotes kbroman/miner, kbroman/Rmaze

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libfftw3-dev libglpk-dev make libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libxml2-dev libx11-dev zlib1g-dev

Repository https://kbroman.r-universe.dev

Date/Publication 2026-06-01 15:02:19 UTC

RemoteUrl https://github.com/kbroman/craft

RemoteRef HEAD

RemoteSha 132b006debf54f7928525b27cb381764439f6656

Contents

`buildBuilding` *Build a building*

Description

Build a building with specified dimensions, with a foundation, walls, floorboards, carpet, and a door

Usage

```
buildBuilding(
  length = 8,
  width = 6,
  height = 5,
  foundation = 1,
  wall = 45,
  floorBoards = 125,
  carpet = 171,
  carpet_style = 11,
  gap = 0,
  player_id = NULL
)
```

Arguments

`length`	Length of the building
`width`	Width of the building
`height`	Height of the building
`foundation`	ID for foundation blocks (1 = stone)
`wall`	ID for wall blocks (45 = brick)
`floorBoards`	ID for floor boards (125 = wood planks)
`carpet`	ID for carpet blocks (171 = carpet)
`carpet_style`	ID for carpet style (11 = blue)
`gap`	ID for gap blocks (0 = air)
`player_id`	Player ID

Value

None.

Author(s)

Felix Ling

`buildDoor` *Builds a door*

Description

Build a door in the wall (or any non-AIR block) that the player is looking at, opening in that same direction. If desired, also puts a pressure plate in front of and behind the door to automatically open it. Alternatively, can specify the coordinates and direction manually.

Usage

```
buildDoor(
  x = NULL,
  y = NULL,
  z = NULL,
  direction = NULL,
  doorBlock = 64,
  pressurePlate = TRUE,
  pressurePlateBlock = 70,
  player_id = NULL
)
```

Arguments

`x`	x-coordinate of where to put the door. If any of `x`, `y`, or `z` are NULL, will calculate from player's position and direction they are facing.
`y`	y-coordinate of where to put the door.
`z`	z-coordinate of where to put the door.
`direction`	The compass direction the door should open in.
`doorBlock`	The material to create the door out of.
`pressurePlate`	Whether to put pressure plates around the door.
`pressurePlateBlock`	The material to create the pressure plate out of.
`player_id`	Player ID

Details

For door blocks, see `mc::find_item("door")`; they include 64 for oak and 71 for iron. Pressure place blocks include 70 for stone and 72 for wood.

Author(s)

Felix Ling

`buildFence` *Build a fence around a square area*

Description

Build a fence around a square area, adding a gate in a random side, and filling the ground underneath to make a uniform height

Usage

```
buildFence(
  length = 8,
  fenceBlock = 85,
  gateBlock = 107,
  foundationBlock = 1,
  player_id = NULL
)
```

Arguments

`length`	Length and width of fence (must be >= 3)
`fenceBlock`	Block ID for fence
`gateBlock`	Block ID for gate in fence
`foundationBlock`	Block ID for foundation (to get everything to a uniform height)
`player_id`	Player ID; fence centered at player's current position

Value

None.

`buildRlogo` *Render R logo in minecraft*

Description

Render the R logo in minecraft

Usage

```
buildRlogo(
  bottomleft,
  height = 80,
  width = NULL,
  dir = c("north", "south", "east", "west"),
  blue_id = 35,
  blue_style = 11,
  gray_id = 35,
  gray_style = 8
)
```

Arguments

bottomleft	Bottom left position (
height	Height of R logo in blocks
width	Width of R logo in blocks (if not provided, determined to preserve aspect ratio
dir	Which direction should the logo go?
blue_id	Block ID for blue blocks
blue_style	Block style for blue blocks
gray_id	Block ID for gray blocks
gray_style	Block style for gray blocks

Examples

```
## Not run:
library(miner)
mc_connect()
pos <- getPlayerPos(getPlayerIds()[1])
buildRlogo(pos + c(5, 10, 5))

# used stained glass
blue_glass <- find_item("Blue Stained Glass")
gray_glass <- find_item("Gray Stained Glass")
buildRlogo(pos + c(5, 10, 5),
           blue_id=blue_glass[2], blue_style=blue_glass[3],
           gray_id=gray_glass[2], gray_style=gray_glass[3])

## End(Not run)
```

---

buildStairs *Builds stairs*

Description

Build stairs up or down from player position in the specified direction. For now, width should be odd so that we can have a central staircase and extend to the right and left of it to widen it.

Usage

```
buildStairs(
  player_id = NULL,
  down = TRUE,
  stopHeight = 0,
  stopOnNotAir = TRUE,
  width = 1,
  stairId = 109,
  blockId = 98,
  blockStyle = 0,
  buildBlock = TRUE
)
```

Arguments

player_id	Player ID
down	Whether to build stairs downward. If false, builds upward
stopHeight	The height at which to stop building.
stopOnNotAir	Whether to stop if about to build where something exists.
width	How wide to build the staircase. MUST BE AN ODD NUMBER
stairId	The block ID for the stairs.
blockId	The block ID for the block underneath each stair. Can set to AIR if you don't want this block.
blockStyle	The block style for the block underneath each stair.
buildBlock	If false, don't put the blocks under the stairs

Details

For stairs block IDs, see `miner::find_item("stairs")`; they include 53 for oak, 67 for cobblestone, and 108 for brick. The corresponding blocks for these materials are 5 for oak, 4 for cobblestone, and 45 for brick

---

clearSpace *Clear out a cube around a player*

Description

Clear out a cube around a player

Usage

```
clearSpace(length, player_id = NULL)
```

Arguments

length	length of each side of the cube to be cleared
player_id	player's entity id

Author(s)

Felix Ling

See Also

miner::setBlocks(), miner::getPlayerPos()

Examples

```
## Not run:
id <- getPlayerIds()[1]
clearSpace(5, id)

## End(Not run)
```

---

cube *Make a cube*

Description

Make a solid or hollow cube.

Usage

```
cube(
  xlen = 10,
  ylen = 10,
  zlen = 10,
  blockid = 1,
  styleid = 0,
  fill = FALSE,
  offset = c(0, -1, 0),
  playerid = miner::getPlayerIds(),
  pos,
  xlim,
  ylim,
  zlim
)
```

Arguments

xlen	integer. Cube length.
ylen	integer. Cube hight.
zlen	integer. Cube depth.
blockid	integer. Block type.
styleid	integer. Block style.
fill	logical. If TRUE then solid. If FALSE then hollow.
offset	vector of length three. How many units to offset the cube from pos.
playerid	integer of length one. Defaults to player id in solo play. Set explicitly in multi play.
pos	vector of length three. Defines the corner of the cube. Defaults to the player's position.
xlim	vector of length two. Defines the lower and upper cuttoff points to truncate the cube.
ylim	vector of length two. Defines the lower and upper cuttoff points to truncate the cube.
zlim	vector of length two. Defines the lower and upper cuttoff points to truncate the cube.

Value

Builds a cube or cuboid with the corner at the player's position. By default, the cube is hollow, but you can also use `fill=TRUE` to create a solid cube. The players position is determined by `miner::getPlayerPos()`. You can you can reposition the cube with the `offset` command. Use `xlim`, `ylim`, `zlim` to truncate the cube to create a fence or wall. This function will return the cube origin.

See Also

`sphere()` to create a sphere.

Examples

```
## Not run:

# Stone cube 10x10x10
cube(pos = c(0, 0, 0))

# Erase cube
cube(blockid = 0, pos = c(0, 0, 0))

# Stone fence
cube(ylim = c(2, 3), offset = c(0, -2, 0))

## End(Not run)
```

de_elsafy *Give a player reverse Elsa powers*

Description

Give a player revsese Elsa powers, to clean up after running `elsafy()`: when she walks over ice, it turns back to water.

Usage

```
de_elsafy(player_id = NULL, water = 8, ice = 174, delay = 0.05)
```

Arguments

`player_id`	Player's entity ID
`water`	Item IDs for water
`ice`	Item ID for ice
`delay`	Delay (in seconds) between calls to the minecraft server

Value

None.

See Also

`elsafy()`

Examples

```
## Not run:
# Runs in an infinite loop; press ctrl-C to stop.
de_elsafy(215)

## End(Not run)
```

drawLine *Draws a line*

Description

Implements Bresenham's line-drawing algorithm in 3D space. Takes two points, where each point is a vector of 3 coordinates, and constructs a line between the two points.

Usage

```
drawLine(p0, p1, id = 1, style = 0)
```

Arguments

p0	vector of the first endpoint
p1	vector of the second endpoint
id	Minecraft block ID to draw the line with
style	Minecraft block style ID to draw the line with (e.g., color)

Value

None

Author(s)

Felix Ling, based on python code in a gist by theJollySin that has since been removed

elsafy *Give a player Elsa powers*

Description

Give a player Elsa powers: when she walks on water, it turns to ice.

Usage

```
elsafy(player_id = NULL, water = c(8, 9), ice = 174, delay = 0.02)
```

Arguments

player_id	Player's entity ID
water	Vector of item IDs for different kinds of water
ice	Item ID for ice
delay	Delay (in seconds) between calls to the minecraft server

Value

None.

See Also

de_elsafy()

Examples

```
## Not run:
# Runs in an infinite loop; press ctrl-C to stop.
elsafy(215)

## End(Not run)
```

`find_items` *Find multiple items by name or ID/style*

Description

Find a set of Minecraft items by name or ID. If querying by ID, the search can also specify item styles.

Usage

```
find_items(name = NULL, id = NULL, style = 0)
```

Arguments

`name`	Vector of character string with the names of Minecraft items (specify either `name` or `id`, not both)
`id`	Vector of numeric or character strings with the ID of a Minecraft item (specify either `name` or `id`, not both)
`style`	Vector of numeric or character string with the style of a Minecraft item (use this argument only if querying by `id` is provided); should have length 1 or the same length as `id`.

Details

If `name` is provided, we first look to see whether there is an exact match to the `name` column in `mc_items`. If there is, we return that row. If not, we don't include the results.

If instead `id` is provided, we return the row with that id and `style==0` (or whatever style was provided).

Value

Data frame with a row for each item found in `mc_items`, provided the matches are unique.

See Also

miner::find_item(), miner::mc_items

Examples

```
flower_names <- c("Chorus Flower", "Peony", "Rose Bush",
    "Lilac", "Sunflower", "Pink Tulip", "White Tulip",
    "Orange Tulip", "Red Tulip", "Oxeye Daisy", "Allium",
    "Dandelion", "Poppy", "Blue Orchid", "Peony")
flowers <- find_items(flower_names)
```

---

font_sets *Font set data*

Description

Font set data used to construct text from blocks within minecraft

Usage

```
data(font_sets)
```

Format

An object of class `list` of length 12.

Details

The dataset is a list of fonts with each font being represented as a list with

- `charset` - a vector of character strings with the characters present in the font
- `png` - a matrix of 0's and 1's grabbed from the PNG file for the font using the package imager.

Source

http://uzebox.org/wiki/index.php?title=Font_Bitmaps

Examples

```
data(font_sets)
```

getBlocksV *getBlocks but taking vector positions*

Description

Determine blocks within a cuboid

Usage

```
getBlocksV(pos0, pos1)
```

Arguments

pos0	Vector (x,y,z) of first position
pos1	Vector (x,y,z) of second position

Value

An 3-D array of integers where each integer gives the ID of the type of a block in the cuboid.

See Also

miner::getBlocks()

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
getBlocksV(p - c(0, 1, 0), p - c(0, 5, 0))

## End(Not run)
```

getBlockV *getBlock but taking a vector position*

Description

Determine block at position (x,y,z)

Usage

```
getBlockV(pos, include_style = TRUE)
```

Arguments

pos	Vector (x,y,z) of position
include_style	A logical value of whether the block's style should also be included in the output (defaults to TRUE).

Value

A numeric vector of length one or two with the type ID and style, if include_style is TRUE, of the block at position (x, y, z). You can use find_item() to find the name of the block type based on this returned ID.

See Also

miner::getBlock()

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
getBlockV(p + c(0, -1, 0))

## End(Not run)
```

---

getHeading *Get player heading*

Description

Get player heading

Usage

```
getHeading(player_id)
```

Arguments

player_id	An integer with the player id

Value

Player heading in degrees

Examples

```
## Not run:
getHeading(myid)

## End(Not run)
```

`getPlayerCompass` *Get player's rotation as compass bearing*

Description

Get the direction that a player is facing as a compass bearing (north, south, east, west, etc.)

Usage

```
getPlayerCompass(player_id = NULL, n_compass_points = c("16", "8", "4"))
```

Arguments

`player_id`	Numeric ID for the player
`n_compass_points`	Number of compass points to use (4, 8, or 16)

Value

A character string representing the player's direct as a compass bearing

Examples

```
## Not run:
library(miner)
getPlayerIds()
getPlayerRotation(355)
getPlayerCompass(355, 8)

## End(Not run)
```

`ice_towers` *Make a random ice tower at hit locations*

Description

Make an ice tower of a random height wherever you hit.

Usage

```
ice_towers(
  player_id,
  block_id = 212,
  block_style = 0,
  max_height = 4,
  delay = 0.2
)
```

Arguments

`player_id`	Player's entity ID
`block_id`	Item ID (id=212 for frosted ice)
`block_style`	Item style
`max_height`	Maximum height of ice tower
`delay`	Delay (in seconds) between calls to the minecraft server

Value

None.

Note

Only right clicks with an iron sword will work.

See Also

`miner::getBlockHits()`

Examples

```
## Not run:
# connect to minecraft
miner::mc_connect()

# Need to use ctrl-c to stop
ice_towers( getPlayerIds()[1] )

## End(Not run)
```

initHeading *Set player heading to match the direction the player is facing*

Description

Set player heading to match the direction the player is facing

Usage

```
initHeading(player_id)
```

Arguments

player_id	An integer with the player id

Examples

```
## Not run:
initHeading(myid)

## End(Not run)
```

lookForward *Check the block type (if any) ahead of the player*

Description

Check the block type (if any) ahead of the player. Look in the direction of the player heading at the specified distance.

Usage

```
lookForward(player_id, distance = 1)
```

Arguments

player_id	An integer with the player id
distance	The distance to look

Examples

```
## Not run:
lookForward(myid)

## End(Not run)
```

`mc_clearplot` *Clear scatterplot*

Description

Remove a scatterplot created with `mc_plot()` by replacing the blocks with air.

Usage

```
mc_clearplot(
  lowerleft,
  x,
  y,
  xlab = "x",
  ylab = "y",
  width = 120,
  height = 120,
  dir = c("east", "west", "north", "south", "up", "down"),
  top = c("up", "north", "south", "east", "west", "down")
)
```

Arguments

`lowerleft`	Vector of length 3, specifying the position of the lower-left corner of the plot.
`x`	Vector of x values
`y`	Vector of y values
`xlab`	x-axis label
`ylab`	y-axis label
`width`	Width of plot in blocks
`height`	Height of plot in blocks
`dir`	Direction the plot will go
`top`	Direction for the top of the plot

Value

None.

See Also

`mc_plot()`

Examples

```
## Not run:
v <- mc_plot(x=iris$Sepal.Length, y=iris$Sepal.Width, group=iris$Species,
             xlab="Sepal.Length", ylab="Sepal.Width")
Sys.sleep(10)
mc_clearplot(v, x=iris$Sepal.Length, y=iris$Sepal.Width,
             xlab="Sepal.Length", ylab="Sepal.Width")

## End(Not run)
```

---

mc_maze *Generates a maze in front of a player*

Description

Generates a maze in front of a player

Usage

```
mc_maze(n = 5, player_id = NULL)
```

Arguments

n	size of the maze
player_id	optional player id

---

mc_mazer *Poll the chat window for maze generator commands and spawn mazes right in front of the player*

Description

Poll the chat window for maze generator commands and spawn mazes right in front of the player

Usage

```
mc_mazer()
```

mc_plot *Scatterplot within Minecraft*

Description

Make a scatterplot within Minecraft

Usage

```
mc_plot(
  lowerleft = miner::getPlayerPos() + c(0, 5, 5),
  x,
  y,
  xlab = "x",
  ylab = "y",
  group = NULL,
  width = 120,
  height = 120,
  dir = c("east", "west", "north", "south", "up", "down"),
  top = c("up", "north", "south", "east", "west", "down"),
 block_colors = list(gray = c(35, 8), white = c(35, 0), black = c(35, 15), colors =
    cbind(35, c(9, 14, 5)))
)
```

Arguments

lowerleft	Vector of length 3, specifying the position of the lower-left corner of the plot.
x	Vector of x values
y	Vector of y values
xlab	x-axis label
ylab	y-axis label
group	Vector of groups for coloring the points
width	Width of plot in blocks
height	Height of plot in blocks
dir	Direction the plot will go
top	Direction for the top of the plot
block_colors	A list of block IDs and styles to denote the colors to be used; needs to contain "gray", "white", "black", and "colors".

Value

Returns the input lowerleft.

See Also

mc_clearplot()

Examples

```
## Not run:
v <- mc_plot(getPlayerPos()+c(0, 5, 5),
             x=iris$Sepal.Length, y=iris$Sepal.Width,
             group=iris$Species,
             xlab="Sepal.Length", ylab="Sepal.Width")

## End(Not run)
```

---

mc_race *Determine who is fastest*

Description

Hold a race, started in the chat, and see who moves the farthest

Usage

```
mc_race(time = 10)
```

Arguments

time	Length of race (in seconds)

Value

Return the player_id of whoever moves the farthest

Examples

```
## Not run:
library(miner)
mc_connect()
whoisfastest()

## End(Not run)
```

mc_Reval *Evaluate R within Minecraft*

Description

Look for chat messages that start with "R " and then evaluate the rest as an R expression.

Usage

```
mc_Reval(player_id = NULL, delay = 1)
```

Arguments

player_id	Player's entity ID
delay	Delay (in seconds) between checks of the chat messages

Details

The function looks at chat messages in Minecraft. For messages that begin "R ", the rest of the message is treated as an R expression and is evaluated.

The default is to evaluate any such R code. If player_id is specified, only messages from that player are considered.

Use **extreme** caution with this; if you don't specify player_id correctly, you could be letting others execute code on your computer and so giving them the opportunity to do bad things.

Value

None.

Examples

```
## Not run:
# runs in an infinite loop; use ctrl-c to stop
mc_Reval(212)
## End(Not run)
```

mc_whoami *Listen for "Who am I"*

Description

Listen for "Who am I" on minecraft chat, and respond with the player's entity ID.

Usage

```
mc_whoami(delay = 1, max_time = Inf)
```

Arguments

delay	Delay (in seconds) between checks of the chat messages
max_time	Maximum amount of time (in seconds) before quitting.

Details

Uses base::grepl(), ignoring case, to look for "who am I" in the chat messages, and prints the player ID of any player who posts a chat message containing that phrase.

Value

A vector of player IDs

Examples

```
## Not run:
#' will run for 15 seconds
mc_whoami(max_time=15)
## End(Not run)
```

moveForward *Move the player in the direction of the player heading*

Description

Move the player in the direction of the player heading

Usage

```
moveForward(player_id, distance = 1)
```

Arguments

player_id	An integer with the player id
distance	The distance to move the player

Examples

```
## Not run:
initHeading(myid)
moveForward(myid, 0.5)

## End(Not run)
```

---

num_guess *Guess a number game*

Description

Play guess-a-number game in the chat

Usage

```
num_guess(max_num = 100, delay = 0.2)
```

Arguments

max_num	Maximum number
delay	Delay (in seconds) between calls to the minecraft server

Value

The correct number

See Also

miner::chatPost(), miner::getChatPosts()b

Examples

```
## Not run:
# connect to minecraft
miner::mc_connect()

# Need to use ctrl-c to stop early
num_guess()

## End(Not run)
```

Rlogo *R logo data*

Description

Matrix corresponding to a version of the R logo

Usage

```
data(Rlogo)
```

Format

An object of class `cimg` (inherits from `imager_array`, `numeric`) of dimension 800 x 700 x 1 x 1.

Details

The dataset is a matrix of pixel colors for a reduced-size version of the R logo, with 1=white, 2=gray, and 3=blue.

Source

https://www.r-project.org/logo/"

Examples

```
data(Rlogo)
image(Rlogo[,ncol(Rlogo):1], col=c("white", "gray", "blue"),
      bty="n", xaxt="n", yaxt="n")
```

setBlocksMix *Place a random mixture of blocks in a cuboid*

Description

Place blocks of a random mixture of types in the cuboid with opposite corners at the positions (x0, y0, z0) and (x1, y1, z1).

Usage

```
setBlocksMix(x0, y0, z0, x1, y1, z1, ids, styles = NULL, prob = NULL)
```

Arguments

x0	A numeric string with north/south position of one corner
y0	A numeric string with height of one corner
z0	A numeric string with east/west position of one corner
x1	A numeric string with north/south position of opposite corner
y1	A numeric string with height of opposite corner
z1	A numeric string with east/west position of opposite corner
ids	Vector of block ids
styles	Vector of block styles (same length as ids)
prob	Probabilities for each block type (same length as ids) If NULL, they are taken as equally probable.

Details

This is like miner::setBlocks() but placing a random mixture of blocks.

Value

None.

Author(s)

Felix Ling

See Also

miner::setBlock(), miner::setBlocks(), setBlocksStyle()

Examples

```
## Not run:
mc_connect()

items <- find_item("stained glass$")
pos <- whereami()
setBlocksMix(pos[1]+2, pos[2], pos[3]+2,
             pos[1]+2, pos[2]+5, pos[3]+2,
             ids=items[,2], styles=items[,3])

## End(Not run)
```

`setBlocksMixV` *setBlocksMix but taking vector positions*

Description

Place a cuboid of blocks of a random mixture of types

Usage

```
setBlocksMixV(pos0, pos1, ids, styles = NULL, prob = NULL)
```

Arguments

`pos0`	Vector (x,y,z) of first position
`pos1`	Vector (x,y,z) of second position
`ids`	Vector of block IDs
`styles`	Vector of block styles
`prob`	Probabilities for each block type

Value

None.

See Also

`setBlocksMix()`

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
setBlocksStyleV(p + c(0, 1, 0), p + c(0, 5, 0), 95, 2)

## End(Not run)
```

`setBlocksStyle` *Place blocks in a cuboid*

Description

Place blocks of a single type (specified by `id`) in the cuboid with opposite corners at the positions (x0, y0, z0) and (x1, y1, z1).

Usage

```
setBlocksStyle(x0, y0, z0, x1, y1, z1, id, style = 0)
```

Arguments

`x0`	A numeric string with north/south position of one corner
`y0`	A numeric string with height of one corner
`z0`	A numeric string with east/west position of one corner
`x1`	A numeric string with north/south position of opposite corner
`y1`	A numeric string with height of opposite corner
`z1`	A numeric string with east/west position of opposite corner
`id`	Block id
`style`	Block style

Details

This is just like `miner::setBlocks()` but with an added `style` argument.

Value

None.

Author(s)

Felix Ling

See Also

`miner::setBlock()`, `miner::setBlocks()`

Examples

```
## Not run:
mc_connect()

item <- find_item("Blue Stained Glass")
pos <- whereami()
setBlocksStyle(pos[1]+2, pos[2], pos[3]+2,
          pos[1]+2, pos[2]+5, pos[3]+2,
          id=item[2], style=item[3])

## End(Not run)
```

`setBlocksStyleV` *setBlocksStyle but taking vector positions*

Description

Place a cuboid of blocks of a single type, allowing a style parameter

Usage

```
setBlocksStyleV(pos0, pos1, id, style)
```

Arguments

`pos0`	Vector (x,y,z) of first position
`pos1`	Vector (x,y,z) of second position
`id`	Block ID
`style`	Block style

Value

None.

See Also

`setBlocksStyle()`

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
setBlocksStyleV(p + c(0, 1, 0), p + c(0, 5, 0), 95, 2)

## End(Not run)
```

setBlocksV *setBlocks but taking vector positions*

Description

Place a cuboid of blocks of a single type

Usage

```
setBlocksV(pos0, pos1, id)
```

Arguments

pos0	Vector (x,y,z) of first position
pos1	Vector (x,y,z) of second position
id	Block ID

Value

None.

See Also

miner::setBlocks()

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
setBlocksV(p + c(0, 1, 0), p + c(0, 5, 0), 46)

## End(Not run)
```

setBlockV *setBlock but taking a vector position*

Description

Place a block at position (x,y,z) by type id

Usage

```
setBlockV(pos, id, style = 0)
```

Arguments

pos	Vector (x,y,z) of position
id	Block ID
style	Block style

Value

None.

See Also

miner::setBlock()

Examples

```
## Not run:
library(miner)
mc_connect()
p <- getPlayerPos()
setBlockV(p + c(0, 5, 0), 46)

## End(Not run)
```

---

setHeading *Set player heading*

Description

Set the direction in which moveForward will move the player. The heading is measured in degrees. A heading of 180 or -180 degreees is towards North or negative z A heading of 0 degreees is towards South or positive z A heading of 270 or -90 degreees is towards East or positive x A heading of 90 degrees is towards West or negative x

Usage

```
setHeading(player_id, new_heading)
```

Arguments

player_id	An integer with the player id
new_heading	A numeric with the direction to head forward

Examples

```
## Not run:
setHeading(myid, 0)

## End(Not run)
```

`setPlayerDirectionV` *setPlayerDirection but taking a vector of positions*

Description

Rotate player to direction (x,y,z) specified as a vector

Usage

```
setPlayerDirectionV(pos, player_id = NULL)
```

Arguments

`pos`	Vector (x,y,z) giving position
`player_id`	Player ID

Value

None.

See Also

`miner::setPlayerDirection()`

Examples

```
## Not run:
library(miner)
mc_connect()
setPlayerDirectionV( c(1, 0, 1) )

## End(Not run)
```

setPlayerPosV *setPlayerPos but taking a vector of positions*

Description

Move player to position (x,y,z) specified as a vector

Usage

```
setPlayerPosV(pos, player_id = NULL, tile = FALSE)
```

Arguments

pos	Vector (x,y,z) giving position
player_id	Player ID
tile	If TRUE, truncation position to integers

Value

None.

See Also

miner::setPlayerPos()

Examples

```
## Not run:
library(miner)
mc_connect()
pos <- getPlayerPos()
setPlayerPosV(pos + c(0, 5, 0))

## End(Not run)
```

sphere *Make a sphere*

Description

Make a solid or hollow sphere with the player at the center.

Usage

```
sphere(
  radius = 15,
  blockid = 20,
  styleid = 0,
  fill = FALSE,
  offset = c(0, 0, 0),
  playerid = miner::getPlayerIds(),
  pos,
  xlim,
  ylim,
  zlim
)
```

Arguments

radius	integer. Specifies the radius of the sphere.
blockid	integer. Block type to be used.
styleid	integer. Block style to be used.
fill	logical. Should the sphere be solid or hollow. Defaults to hollow.
offset	vector of length three. Defines where to place the sphere with respect to pos.
playerid	integer of length one. Defaults to player id in solo play. Set explicitly in multi play.
pos	vector of length three. Defines the center of the sphere. Defaults to the player id position.
xlim	vector of length two. Defines the lower and upper cuttoff points to truncate the sphere.
ylim	vector of length two. Defines the lower and upper cuttoff points to truncate the sphere.
zlim	vector of length two. Defines the lower and upper cuttoff points to truncate the sphere.

Value

Builds a sphere (or a truncated sphere) around the players current position. By default, the sphere is hollow, but you can also use `fill=TRUE` to create a solid sphere. The players position is determined by `miner::getPlayerPos()`. You can you can reposition the sphere with the `offset` command. Use `xlim`, `ylim`, `zlim` to truncate the sphere. This function will return the sphere's origin.

See Also

`cube()` to create a cube.

Examples

```
## Not run:

# Hollow glass sphere of size 15
sphere(pos = c(0, 0, 0))

# Erase sphere
sphere(blockid = 0, pos = c(0, 0, 0))

# Glass dome
sphere(ylim = c(0, 15))

## End(Not run)
```

`turnLeft` *Rotate the player heading to the left*

Description

Use a positive angle value to rotate to the left. Use a negative angle to rotate to the right. The player does not turn in the game; rather, the heading for subsequent movement commands is rotated.

Usage

```
turnLeft(player_id, angle = 90)
```

Arguments

`player_id`	An integer with the player id
`angle`	The number of degrees to rotate left

Examples

```
## Not run:
turnLeft(myid)

## End(Not run)
```

whereami *Get rounded player position*

Description

Get rounded player position

Usage

```
whereami(player_id = NULL, tile = FALSE)
```

Arguments

player_id	Integer giving the ID of a player
tile	If TRUE, truncate the result to integers

Details

This is just like `miner::getPlayerPos()` but with `tile=TRUE` being the default.

Author(s)

Felix Ling

See Also

`miner::getPlayerPos()`

Examples

```
## Not run:
mc_connect()
whereami()

## End(Not run)
```

write_text *Write text in minecraft*

Description

Write some text in minecraft, using blocks

Usage

```
write_text(
  text,
  lowerleft,
  font = c("4x5", "4x6", "4x8", "6x6", "8x6", "6x8", "8x8", "4x12", "6x12", "8x12",
    "8x16", "16x16"),
  id = 1,
  style = 0,
  dir = c("north", "south", "east", "west", "up", "down"),
  top = c("up", "east", "west", "north", "south", "down")
)
```

Arguments

text	A character string to be written in blocks
lowerleft	Vector of length 3 representing the location for the lower-left corner for the text.
font	Font size to use
id	Block type to use to write the text
style	Block style to use to write the text
dir	Direction the text should run
top	Direction that the top of the text should point

Value

None

Examples

```
## Not run:
library(miner)
v <- getPlayerPos()
write_text("Hello!", v+c(5,3,0), font="4x8")

## End(Not run)
```

Index