Package 'chess'

October 12, 2022

Title Read, Write, Create and Explore Chess Games

Version 1.0.1

Description This is an opinionated wrapper around the python-chess package. It allows users to read and write PGN files as well as create and explore game trees such as the ones seen in chess books.

License GPL-3

URL https://github.com/curso-r/chess

BugReports https://github.com/curso-r/chess/issues

Depends R (>= 2.10)

Imports cli, magrittr, purrr, reticulate, rsvg

Suggests covr, graphics, knitr, png, rmarkdown, testthat

VignetteBuilder knitr

Config/reticulate list(packages = list(list(package = ``python-chess'', pip = TRUE)))

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author C. Lente [aut, cre]

Maintainer C. Lente <clente@curso-r.com>

Repository CRAN

Date/Publication 2020-12-04 22:20:02 UTC

R topics documented:

back . . . 2
board_color . . . 3
board_is . . . 4
board_move . . . 5
board_to_string . . . 5
fen . . . 6
forward . . . 6
game . . . 7
glyph_to_nag . . . 7
halfmove_clock . . . 8
install_chess . . . 8
line . . . 9
move . . . 9
moves . . . 10
move_ . . . 10
move_number . . . 11
nag . . . 11
note . . . 12
parse_move . . . 12
pgn . . . 13
play . . . 13
plot.chess.pgn.GameNode . . . 14
ply_number . . . 14
print.chess.Board . . . 15
print.chess.pgn.GameNode . . . 15
print.chess.pgn.Variations . . . 16
read_game . . . 16
result . . . 17
root . . . 17
turn . . . 18
variation . . . 18
variations . . . 19
write_game . . . 19
write_svg . . . 20

Index **21**

---

back *Go back in the game tree, reverting the last move from current branch*

Description

Go back in the game tree, reverting the last move from current branch

Usage

```
back(game, steps = 1)
```

Arguments

game	A game node
steps	How many steps (half-turns) to go back

Value

A game node

`board_color` *Get information about the current board given a color*

Description

Get information about the current board given a color

Usage

```
has_insufficient_material(game, color)

has_castling_rights(game, color)

has_kingside_castling_rights(game, color)

has_queenside_castling_rights(game, color)
```

Arguments

game	A game node
color	Color to use (`TRUE` is White and `FALSE` is Black)

Value

A boolean

board_is *Get information about the current board*

Description

Get information about the current board

Usage

```
is_checkmate(game)

is_check(game)

is_game_over(game)

is_stalemate(game)

is_insufficient_material(game)

is_seventyfive_moves(game)

is_fivefold_repetition(game)

is_repetition(game, count = 3)

can_claim_draw(game)

can_claim_fifty_moves(game)

can_claim_threefold_repetition(game)

has_en_passant(game)
```

Arguments

game	A game node
count	Number of moves to count for repetition

Value

A boolean

board_move *Get information about the current board given a move*

Description

Get information about the current board given a move

Usage

```
gives_check(game, move, notation = c("san", "uci", "xboard"))

is_en_passant(game, move, notation = c("san", "uci", "xboard"))

is_capture(game, move, notation = c("san", "uci", "xboard"))

is_zeroing(game, move, notation = c("san", "uci", "xboard"))

is_irreversible(game, move, notation = c("san", "uci", "xboard"))

is_castling(game, move, notation = c("san", "uci", "xboard"))

is_kingside_castling(game, move, notation = c("san", "uci", "xboard"))

is_queenside_castling(game, move, notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
move	Move to consider
notation	Notation used for move

Value

A boolean

board_to_string *Convert a board to either unicode or ASCII string*

Description

Convert a board to either unicode or ASCII string

Usage

```
board_to_string(x, unicode = FALSE, invert_color = FALSE, empty_square = ".")
```

Arguments

x	A board
unicode	Use unicode characters?
invert_color	Invert piece color? Useful for white text on dark background.
empty_square	Character used for empty square

Value

A string

---

fen *Get FEN representation of board*

Description

Get FEN representation of board

Usage

```
fen(game)
```

Arguments

game	A game node

Value

A string

---

forward *Advance in the game tree, playing next move from current branch*

Description

Advance in the game tree, playing next move from current branch

Usage

```
forward(game, steps = 1)
```

Arguments

game	A game node
steps	How many steps (half-turns) to advance

Value

A game node

game *Create a new game*

Description

A game is a tree with nodes, where each node represents the board after a move and each branch represents a variation of the game (not to be confused with a variant of chess). This tree mirrors the PGN of the game.

To explore a game, an object of this class supports `print()`, `plot()`, `str()`, `fen()`, `pgn()` and more.

Usage

```
game(headers = NULL, fen = NULL)
```

Arguments

`headers`	A named list like `list("Header1" = "Value1", ...)`
`fen`	FEN representing the starting position of the board

Value

A game root node

glyph_to_nag *Convert glyph to NAG*

Description

Convert glyph to NAG

Usage

```
glyph_to_nag(glyph)
```

Arguments

`glyph`	A game node

Value

An integer

halfmove_clock *Get number of half-moves since the last capture or pawn move*

Description

Get number of half-moves since the last capture or pawn move

Usage

```
halfmove_clock(game)
```

Arguments

game	A game node

Value

An integer

install_chess *Install python-chess*

Description

Install the python library used as the backbone of this package. You can pass arguments on to `reticulate::py_install()`, but `python-chess` needs `python_version = "3.8"` and `pip = TRUE`.

Usage

```
install_chess(method = "auto", conda = "auto", ...)
```

Arguments

method	Installation method
conda	The path to a conda executable
...	Other arguments passed on to `reticulate::py_install()`

line *Branch game with next move*

Description

Branch game with next move

Usage

```
line(game, moves, notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
moves	Vector of one or more description of moves
notation	Notation used for moves

Value

A game node

move *Make moves and create variations*

Description

Adding moves to a game works roughly in the same way as PGN. Strings are added as single moves, and lists are added as variations (siblings) to the last move made. After adding moves, the game node returned corresponds to the last move of the mainline. See `vignette("chess")` for more information.

Usage

```
move(game, ..., notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
...	Sequence of moves (lists are converted to a variation the same way parentheses work in PGN)
notation	Notation used for moves (san, uci, or xboard)

Value

A game node

moves — *Get all legal moves available*

Description

Get all legal moves available

Usage

```
moves(game)
```

Arguments

game	A game node

Value

A vector of strings

move_ — *Make moves and create variations*

Description

Make moves and create variations

Usage

```
move_(game, moves, notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
moves	List of moves
notation	Notation used for moves

Value

A game node

move_number *Get number of move*

Description

Get number of move

Usage

```
move_number(game)
```

Arguments

game	A game node

Value

An integer

nag *Parse Numeric Annotation Glyph (NAG) of a move*

Description

Parse Numeric Annotation Glyph (NAG) of a move

Usage

```
nag(game)
```

Arguments

game	A game node

Value

A string

note *Get comment for a move*

Description

Get comment for a move

Usage

```
note(game)
```

Arguments

game	A game node

Value

A string

parse_move *Parse move in context*

Description

Parse move in context

Usage

```
parse_move(game, moves, notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
moves	A move string
notation	Notation used for move

Value

A move object

pgn *Get PGN for node of a game*

Description

Get PGN for node of a game

Usage

```
pgn(game)
```

Arguments

game	A game node

Value

A string

play *Move a piece on the board*

Description

Move a piece on the board

Usage

```
play(game, moves, notation = c("san", "uci", "xboard"))
```

Arguments

game	A game node
moves	Vector of one or more description of moves
notation	Notation used for moves

Value

A game node

plot.chess.pgn.GameNode *Plot rendering of the board*

Description

Plot rendering of the board

Usage

```
## S3 method for class 'chess.pgn.GameNode'
plot(x, ...)
```

Arguments

x	A game node
...	Not used

ply_number *Get number of ply*

Description

Get number of ply

Usage

```
ply_number(game)
```

Arguments

game	A game node

Value

An integer

print.chess.Board *Print board*

Description

Print board

Usage

```
## S3 method for class 'chess.Board'
print(x, unicode = FALSE, invert_color = FALSE, empty_square = ".", ...)
```

Arguments

x	A game board
unicode	Use unicode characters?
invert_color	Invert piece color? Useful for white text on dark background.
empty_square	Character used for empty square
...	Not used

print.chess.pgn.GameNode *Print game node*

Description

Print game node

Usage

```
## S3 method for class 'chess.pgn.GameNode'
print(x, unicode = FALSE, invert_color = FALSE, empty_square = ".", ...)
```

Arguments

x	A game node
unicode	Use unicode characters?
invert_color	Invert piece color? Useful for white text on dark background.
empty_square	Character used for empty square
...	Not used

print.chess.pgn.Variations *Print a list of variations*

Description

Print a list of variations

Usage

```
## S3 method for class 'chess.pgn.Variations'
print(x, unicode = FALSE, invert_color = FALSE, empty_square = ".", ...)
```

Arguments

x	A game node
unicode	Use unicode characters?
invert_color	Invert piece color? Useful for white text on dark background
empty_square	Character used for empty square
...	Not used

read_game *Read a game from a PGN*

Description

Read a .pgn file with one or more annotated games; if there is more than 1 game in the file, a list is returned (which you can access with [[[()]). Some sample .pgn files are included in the package. See vignette("games") for more information.

Usage

```
read_game(file, n_max = Inf)
```

Arguments

file	File or connection to read from
n_max	Maximum number of games to read

Value

A game node or list of game nodes

result *Get result of the game ("*" if it hasn't ended)*

Description

Get result of the game ("*" if it hasn't ended)

Usage

```
result(game)
```

Arguments

game	Any node of a game

Value

A string

root *Get the root node of a game*

Description

Get the root node of a game

Usage

```
root(game)
```

Arguments

game	A game node

Value

A game node

turn *Get whose turn it is*

Description

Get whose turn it is

Usage

```
turn(game)
```

Arguments

game	A game node

Value

A boolean (TRUE is White and FALSE is Black)

variation *Follow variation of a move, playing its first move*

Description

Follow variation of a move, playing its first move

Usage

```
variation(game, id = 1)
```

Arguments

game	A game node
id	Index of variation (1 is the current branch)

Value

A game node

variations *Get all variations for next move (the children of current node)*

Description

Get all variations for next move (the children of current node)

Usage

```
variations(game)
```

Arguments

game	A game node

Value

A list of games nodes

write_game *Save a game as an PGN*

Description

Save a game as an PGN

Usage

```
write_game(x, file)
```

Arguments

x	Any node of a game
file	File or connection to write to

`write_svg` *Save an SVG with rendering of the board*

Description

Save an SVG with rendering of the board

Usage

```
write_svg(x, file)
```

Arguments

`x`	A game node
`file`	File or connection to write to

Index

back, 2
board_color, 3
board_is, 4
board_move, 5
board_to_string, 5

can_claim_draw *(*board_is*)*, 4
can_claim_fifty_moves *(*board_is*)*, 4
can_claim_threefold_repetition *(*board_is*)*, 4

fen, 6
fen(), *7*
forward, 6

game, 7
gives_check *(*board_move*)*, 5
glyph_to_nag, 7

halfmove_clock, 8
has_castling_rights *(*board_color*)*, 3
has_en_passant *(*board_is*)*, 4
has_insufficient_material *(*board_color*)*, 3
has_kingside_castling_rights *(*board_color*)*, 3
has_queenside_castling_rights *(*board_color*)*, 3

install_chess, 8
is_capture *(*board_move*)*, 5
is_castling *(*board_move*)*, 5
is_check *(*board_is*)*, 4
is_checkmate *(*board_is*)*, 4
is_en_passant *(*board_move*)*, 5
is_fivefold_repetition *(*board_is*)*, 4
is_game_over *(*board_is*)*, 4
is_insufficient_material *(*board_is*)*, 4
is_irreversible *(*board_move*)*, 5
is_kingside_castling *(*board_move*)*, 5
is_queenside_castling *(*board_move*)*, 5
is_repetition *(*board_is*)*, 4
is_seventyfive_moves *(*board_is*)*, 4
is_stalemate *(*board_is*)*, 4
is_zeroing *(*board_move*)*, 5

line, 9

move, 9
move_, 10
move_number, 11
moves, 10

nag, 11
note, 12

parse_move, 12
pgn, 13
pgn(), *7*
play, 13
plot(), *7*
plot.chess.pgn.GameNode, 14
ply_number, 14
print(), *7*
print.chess.Board, 15
print.chess.pgn.GameNode, 15
print.chess.pgn.Variations, 16

read_game, 16
result, 17
reticulate::py_install(), *8*
root, 17

str(), *7*

turn, 18

variation, 18
variations, 19

write_game, 19
write_svg, 20