

# Package ‘chess2plyrs’

May 8, 2026

**Type** Package

**Title** Chess Game Creation and Tools

**Version** 0.3.0

**Description** A chess program which allows the user to create a game, add moves, check for legal moves and game result, plot the board, take back, read and write FEN (Forsyth–Edwards Notation). A basic chess engine based on minimax is implemented.

**Depends** R (>= 4.3.0)

**Imports** stats, ggplot2 (>= 3.4.4)

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Luigi Annicchiarico [cre, aut]

**Maintainer** Luigi Annicchiarico <luigi.annic@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-06-02 08:50:09 UTC

## Contents

all_possibilities . . . . .	2
chessplot . . . . .	2
chesstools . . . . .	3
chess_move . . . . .	3
engine1 . . . . .	4
engine2 . . . . .	4

game_result . . . . .	5
get_minimax_move . . . . .	5
legalmoves . . . . .	6
moves_scoresheet . . . . .	6
newgame . . . . .	7
random_mover . . . . .	7
readfen . . . . .	8
takeback . . . . .	8
writefen . . . . .	9

<b>Index</b>	<b>10</b>
--------------	-----------

---

all_possibilities	<i>all_possibilities</i>
-------------------	--------------------------

---

### Description

finds all legal moves for the player moving, and all the squares under attack from the opponent pieces.

### Usage

```
all_possibilities(game)
```

### Arguments

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
------	---

### Value

all chess possibilities

---

chessplot	<i>chessplot</i>
-----------	------------------

---

### Description

plots the current position

### Usage

```
chessplot(game, style = 1)
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
style	font style. 1: chess pieces according to unicode; 2: chess labels

**Value**

plot

---

chesstools	<i>chesstools</i>
------------	-------------------

---

**Description**

All tools used for letting the chess program work.

**Usage**

chesstools

**Format**

An object of class list

---

chess_move	<i>chess_move</i>
------------	-------------------

---

**Description**

Takes in input a move, evaluates whether it is legal, and if it is, then the game is updated

**Usage**

```
chess_move(game, piece, initialposition = "", finalposition = "")
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
piece	letter indicating the piece to be moved (p, N, B, R, Q, K)
initialposition	initial square of the piece
finalposition	destination square

**Value**

makes move

**Examples**

```
newgame() |>
chess_move("N", "g1", "f3")
```

---

engine1	<i>engine1</i>
---------	----------------

---

**Description**

engine which chooses minimax between legal moves

**Usage**

```
engine1(game, depth)
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
depth	depth of the minimax. depth of 1 and 2 are fairly rapid.

**Value**

game with new move done

---

engine2	<i>engine2</i>
---------	----------------

---

**Description**

engine which chooses minimax between legal moves, with alpha beta pruning

**Usage**

```
engine2(game, depth)
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
depth	depth of the minimax. depth of 1 and 2 are fairly rapid.

**Value**

game with new move done

---

*game\_result*                      *game\_result*

---

**Description**

This function tells if the game is still ongoing, or if a checkmate or stalemate are on the board

**Usage**

`game_result(game)`

**Arguments**

`game`                      chess game object (i.e., a list with elements board, turn, history, and fen\_history as created by newgame function)

**Value**

a message

---

*get\_minimax\_move*                      *get\_minimax\_move*

---

**Description**

minimax engine

**Usage**

`get_minimax_move(game, depth)`

**Arguments**

`game`                      chess game object (i.e., a list with elements board, turn, history, and fen\_history as created by newgame function)

`depth`                      algorithm depth

**Value**

minimax engine

`legalmoves`*legalmoves*

---

**Description**

lists legal moves

**Usage**

```
legalmoves(game)
```

**Arguments**

<code>game</code>	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
-------------------	---

**Value**

character vector

**Examples**

```
newgame() |>  
legalmoves()
```

---

`moves_scoresheet`*moves\_scoresheet*

---

**Description**

Creates move scorelist, in scientific notation

**Usage**

```
moves_scoresheet(game, shortnotation = TRUE)
```

**Arguments**

<code>game</code>	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
<code>shortnotation</code>	Use short scientific notation? TRUE is the default

**Value**

moves scoresheet

**Examples**

```
g <- newgame() |>
  chess_move("p", "e2", "e4") |>
  chess_move("p", "e7", "e5") |>
  chess_move("N", "g1", "f3") |>
  chess_move("N", "b8", "c6") |>
  chess_move("B", "f1", "b5") |>
  chess_move("N", "g8", "f6") |>
  chess_move("K", "e1", "0-0") |>
  chess_move("N", "f6", "e4")
moves_scoresheet(g)
```

---

newgame	<i>newgame</i>
---------	----------------

---

**Description**

sets up a new chess game

**Usage**

```
newgame()
```

**Value**

new game

---

random_mover	<i>random_mover</i>
--------------	---------------------

---

**Description**

engine which chooses randomly between legal moves

**Usage**

```
random_mover(game)
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
------	---

**Value**

game with new move done

readfen                      *readfen*

---

**Description**

read fen (Forsyth–Edwards Notation) notation

**Usage**

readfen(fenstring)

**Arguments**

fenstring            fen string (pieces and turn)

**Value**

board and turn

**Examples**

```
readfen("r3kb1r/pp1nqppp/4bp2/2p5/3P4/1B3N2/PPP1QPPP/R1B1K2R w")
```

---

takeback                      *takeback*

---

**Description**

takeback

**Usage**

takeback(game)

**Arguments**

game                      chess game object (i.e., a list with elements board, turn, history, and fen\_history as created by newgame function)

**Value**

game (last move being deleted)

---

writefen	<i>writefen</i>
----------	-----------------

---

**Description**

write fen (Forsyth–Edwards Notation) notation

**Usage**

```
writefen(game, cb = NULL, tb = NULL, cb_tb_insteadof_game = FALSE)
```

**Arguments**

game	chess game object (i.e., a list with elements board, turn, history, and fen_history as created by newgame function)
cb	chess board if cb_tf_insteadof_game set to TRUE
tb	turn if cb_tf_insteadof_game set to TRUE
cb_tb_insteadof_game	if FALSE, uses game to create fen, if TRUE it uses cb and tb

**Value**

fen

# Index

## \* list

- chesstools, 3
- all\_possibilities, 2
- chess\_move, 3
- chessplot, 2
- chesstools, 3
- engine1, 4
- engine2, 4
- game\_result, 5
- get\_minimax\_move, 5
- legalmoves, 6
- moves\_scoresheet, 6
- newgame, 7
- random\_mover, 7
- readfen, 8
- takeback, 8
- writfen, 9