Package ‘ply’

May 9, 2026

Type Package

Title Bitboard Chess Engine

Version 0.1.0

Author Qusai Jouda [aut, cre]

Maintainer Qusai Jouda <jouda.qusai@gmail.com>

Description A fully legal chess move generator and game engine implemented in C++17 via 'Rcpp'. Provides FEN (Forsyth-Edwards Notation) parsing, PGN (Portable Game Notation) replay, position feature enrichment, and a multi-game registry backed by a bitboard representation.

License MIT + file LICENSE

Imports Rcpp (>= 1.0.0)

LinkingTo Rcpp

Suggests testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-03-30 09:20:02 UTC

Contents

`ply_enrich_batch` *Enrich a batch of positions with chess features*

Description

Returns a `data.frame` with one row per (FEN, UCI-move) pair and columns covering captures, checks, castling, promotion, material balance, pawn structure, mobility, king safety, and pin count.

Usage

```
ply_enrich_batch(fens, uci_moves)
```

Arguments

`fens`	Character vector of FEN strings.
`uci_moves`	Character vector of UCI move strings parallel to `fens`. Must be the same length.

Value

A `data.frame` with one row per position and columns: `is_capture`, `is_castling`, `is_promotion`, `is_en_passant`, `gives_check`, `gives_discovered_check`, `in_check`, `material_bal`, `num_pieces`, `legal_move_count`, `moved_piece_value`, `captured_piece_value`, `is_checkmate`, `is_stalemate`, `num_captures_avail`, `num_checks_avail`, `can_castle`, `max_capture_value`, `king_safety_own`, `king_safety_opp`, `mob_pawn`, `mob_knight`, `mob_bishop`, `mob_rook`, `mob_queen`, `mob_king`, `doubled_pawns`, `isolated_pawns`, `passed_pawns`, `pin_count`, `en_passant_avail`, `promotion_available`.

Examples

```
start <- "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
df <- ply_enrich_batch(start, "e2e4")
df$is_capture     # FALSE
df$legal_move_count  # 20
```

---

ply_fen_parse *Parse a FEN string into a ChessState list*

Description

Parse a FEN string into a ChessState list

Usage

```
ply_fen_parse(fen)
```

Arguments

fen	A FEN string.

Value

A named list of class `ChessState` representing the board state.

Examples

```
state <- ply_fen_parse("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1")
state$sideToMove  # 0 = white
```

---

ply_fen_serialize *Convert a ChessState list back to a FEN string*

Description

Convert a ChessState list back to a FEN string

Usage

```
ply_fen_serialize(state)
```

Arguments

state	A `ChessState` list from `ply_fen_parse`.

Value

A FEN string.

Examples

```
state <- ply_game_init()
ply_fen_serialize(state)
```

`ply_game_accept_draw` *Accept an outstanding draw offer in a managed game*

Description

Accept an outstanding draw offer in a managed game

Usage

```
ply_game_accept_draw(game_id, player)
```

Arguments

`game_id`	Integer game id.
`player`	Player name — must be the player who did *not* offer.

Value

`TRUE` on success; `FALSE` if no offer is pending or wrong player.

See Also

`ply_game_offer_draw`

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "e2e4")
ply_game_move(id, "bob",   "e7e5")
ply_game_offer_draw(id, "alice")
ply_game_accept_draw(id, "bob")    # TRUE — game ends as Draw
```

ply_game_cancel *Cancel a waiting managed game*

Description

Only the game creator can cancel, and only while the game is still in *Waiting* status (i.e.\ the opponent has not yet joined).

Usage

```
ply_game_cancel(game_id, player)
```

Arguments

game_id	Integer game id.
player	Creator's player name.

Value

TRUE on success; FALSE if not the creator or game is already active.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")  # Waiting — no opponent yet
ply_game_cancel(id, "alice") # TRUE
ply_game_info(id)$termination  # 10 = Cancelled
```

ply_game_count *Return the number of games currently held in the registry*

Description

Return the number of games currently held in the registry

Usage

```
ply_game_count()
```

Value

A non-negative integer.

See Also

ply_game_new, ply_game_reset_registry

Examples

```
ply_game_reset_registry()
ply_game_count()        # 0
ply_game_new("alice")
ply_game_count()        # 1
```

`ply_game_fen` *Get the current FEN for a managed game*

Description

Get the current FEN for a managed game

Usage

```
ply_game_fen(game_id)
```

Arguments

`game_id`	Integer game id.

Value

A FEN string.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "d2d4")
ply_game_fen(id)  # FEN with white pawn on d4
```

`ply_game_history` *Get the full ply history of a managed game as UCI strings*

Description

Get the full ply history of a managed game as UCI strings

Usage

```
ply_game_history(game_id)
```

Arguments

`game_id`	Integer game id.

Value

A character vector of UCI move strings, one per ply played.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "e2e4")
ply_game_move(id, "bob",   "c7c5")
ply_game_history(id)  # c("e2e4", "c7c5")
```

---

ply_game_info *Get metadata for a managed game*

Description

Get metadata for a managed game

Usage

```
ply_game_info(game_id)
```

Arguments

game_id	Integer game id.

Value

A named list with fields `white`, `black`, `status` (`0`=Waiting, `1`=Active, `2`=Finished), `result` (`0`=InProgress, `1`=WhiteWins, `2`=BlackWins, `3`=Draw), `winner`, `termination`, `fen`, and `plyCount`.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
info <- ply_game_info(id)
info$white   # "alice"
info$status  # 1 (Active)
```

ply_game_init *Return a ChessState list at the standard starting position*

Description

Stateless helper that does not interact with the game registry. To create a managed game use ply_game_new.

Usage

```
ply_game_init()
```

Value

A named list of class ChessState representing the starting chess position.

Examples

```
state <- ply_game_init()
state$sideToMove  # 0 = white to move
state$fullMoveNumber  # 1
```

ply_game_join *Join an existing game as the opposing player*

Description

Join an existing game as the opposing player

Usage

```
ply_game_join(game_id, player)
```

Arguments

game_id	Integer game id.
player	Player name string.

Value

TRUE on success.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")          # TRUE — game is now Active
ply_game_info(id)$status      # 1
```

ply_game_move *Make a move in a managed game by UCI string*

Description

Make a move in a managed game by UCI string

Usage

```
ply_game_move(game_id, player, uci, settle = TRUE)
```

Arguments

game_id	Integer game id.
player	Player name string.
uci	UCI move string (e.g. "e2e4").
settle	If TRUE (default) the engine checks for checkmate and stalemate after the move and marks the game finished if detected. In FULL_FIDE_RULES mode settlement always runs regardless.

Value

TRUE on success.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "e2e4")  # TRUE
ply_game_move(id, "bob",   "e7e5")  # TRUE
ply_game_history(id)           # c("e2e4", "e7e5")
```

ply_game_new *Create a new managed game*

Description

Adds a game to the global registry without resetting existing games. Use ply_game_reset_registry first if you want a clean slate.

Usage

```
ply_game_new(creator, mode = 0L, time_limit = 600)
```

Arguments

creator	Name of the creating player.
mode	Settlement mode: 0 = casual (default), 1 = full FIDE rules (auto-detects repetition and insufficient material).
time_limit	Ply time limit in seconds (0 = no limit, default 600).

Value

Integer game id.

See Also

ply_game_join, ply_game_move, ply_game_reset_registry

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_count()  # 1
```

---

ply_game_new_from_fen *Create a new managed game from a custom starting position*

Description

Create a new managed game from a custom starting position

Usage

```
ply_game_new_from_fen(creator, fen, mode = 0L, time_limit = 600)
```

Arguments

creator	Name of the creating player.
fen	A FEN string for the starting position.
mode	Settlement mode: 0 = casual (default), 1 = full FIDE rules.
time_limit	Ply time limit in seconds (0 = no limit, default 600).

Value

Integer game id.

See Also

ply_game_new

Examples

```
ply_game_reset_registry()
fen <- "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1"
id  <- ply_game_new_from_fen("alice", fen)
ply_game_join(id, "bob")
ply_game_fen(id)  # starts after 1.e4
```

`ply_game_offer_draw` *Offer or toggle a draw in a managed game*

Description

Records a draw offer from `player`. If the opponent has already offered a draw, this call accepts it and concludes the game as a draw by agreement. A player can cancel their own outstanding offer by calling this again with the same `player` name.

Usage

```
ply_game_offer_draw(game_id, player)
```

Arguments

`game_id`	Integer game id.
`player`	Player name string.

Value

`TRUE` on success.

See Also

`ply_game_accept_draw`

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "e2e4")
ply_game_move(id, "bob",   "e7e5")
ply_game_offer_draw(id, "alice")   # alice offers a draw
ply_game_accept_draw(id, "bob")    # bob accepts
ply_game_info(id)$result      # 3 = Draw
```

ply_game_reset_registry *Reset (clear) the global game registry*

Description

Destroys all managed games. Game ids from before the reset become invalid. Subsequent `ply_game_count` calls return 0.

Usage

```
ply_game_reset_registry()
```

Value

Invisibly `NULL`.

See Also

`ply_game_new`, `ply_game_count`

Examples

```
ply_game_new("alice")
ply_game_reset_registry()
ply_game_count()  # 0
```

ply_game_resign *Resign a managed game*

Description

Resign a managed game

Usage

```
ply_game_resign(game_id, player)
```

Arguments

`game_id`	Integer game id.
`player`	Resigning player's name.

Value

`TRUE` on success.

Examples

```
ply_game_reset_registry()
id <- ply_game_new("alice")
ply_game_join(id, "bob")
ply_game_move(id, "alice", "e2e4")
ply_game_resign(id, "bob")          # bob resigns; alice wins
ply_game_info(id)$result       # 1 = WhiteWins
```

---

ply_hash *Compute a Zobrist-style position hash*

Description

Compute a Zobrist-style position hash

Usage

```
ply_hash(state)
```

Arguments

state	A ChessState list.

Value

A 16-character lowercase hex string uniquely identifying the position.

Examples

```
h <- ply_hash(ply_game_init())
nchar(h)  # 16
```

---

ply_in_check *Test whether a side is in check*

Description

Test whether a side is in check

Usage

```
ply_in_check(state, color = state$sideToMove)
```

Arguments

state	A ChessState list.
color	Integer color: 0 = white, 1 = black (default: side to move).

Value

TRUE if the specified side is in check.

Examples

```
ply_in_check(ply_game_init())          # FALSE at the start
ply_in_check(ply_game_init(), color = 0L)  # white not in check
```

`ply_is_checkmate` *Test whether the position is checkmate*

Description

Test whether the position is checkmate

Usage

```
ply_is_checkmate(state)
```

Arguments

`state`	A `ChessState` list.

Value

TRUE if the side to move is checkmated.

Examples

```
# Scholar's Mate — black is checkmated
s <- ply_fen_parse(
  "r1bqkb1r/pppp1Qpp/2n2n2/4p3/2B1P3/8/PPPP1PPP/RNB1K1NR b KQkq - 0 4"
)
ply_is_checkmate(s)  # TRUE
```

ply_is_insufficient_material *Test whether the position is a draw by insufficient material*

Description

Test whether the position is a draw by insufficient material

Usage

```
ply_is_insufficient_material(state)
```

Arguments

state	A ChessState list.

Value

TRUE if neither side can force checkmate.

Examples

```
# Only kings remain
s <- ply_fen_parse("8/8/8/8/8/8/8/K6k w - - 0 1")
ply_is_insufficient_material(s)  # TRUE
```

ply_is_stalemate *Test whether the position is stalemate*

Description

Test whether the position is stalemate

Usage

```
ply_is_stalemate(state)
```

Arguments

state	A ChessState list.

Value

TRUE if the side to move is stalemated.

Examples

```
# Black king a8, white queen c7, white king b6 — stalemate
s <- ply_fen_parse("k7/2Q5/1K6/8/8/8/8/8 b - - 0 1")
ply_is_stalemate(s)  # TRUE
```

`ply_legal_moves` *List all legal moves from a position as UCI strings*

Description

List all legal moves from a position as UCI strings

Usage

```
ply_legal_moves(state)
```

Arguments

`state`	A `ChessState` list.

Value

A character vector of UCI move strings.

Examples

```
moves <- ply_legal_moves(ply_game_init())
length(moves)  # 20 from the starting position
```

`ply_move_apply` *Apply a UCI move string to a position*

Description

Apply a UCI move string to a position

Usage

```
ply_move_apply(state, uci)
```

Arguments

`state`	A `ChessState` list.
`uci`	A UCI move string such as `"e2e4"`.

Value

The new `ChessState` list after the move.

Examples

```
s1 <- ply_game_init()
s2 <- ply_move_apply(s1, "e2e4")
s2$sideToMove  # 1 = black to move
```

---

ply_pgn_extract_movetext *Extract clean SAN movetext tokens from a PGN game block*

Description

Strips tags, comments, variations, NAGs, and result strings.

Usage

```
ply_pgn_extract_movetext(game_text)
```

Arguments

`game_text`	A character string containing one PGN game block.

Value

A single trimmed character string of space-separated SAN tokens.

Examples

```
g <- '[Event "Test"]\n\n1.e4 {Best by test} e5 2.Nf3 Nc6 1-0'
ply_pgn_extract_movetext(g)  # "e4 e5 Nf3 Nc6"
```

---

ply_pgn_load_games *Load a PGN file and return a data.frame of game-level metadata*

Description

Returns a `data.frame` with columns `Event`, `White`, `Black`, `Result`, `ECO`, and `Plys` (number of half-moves).

Usage

```
ply_pgn_load_games(pgn_path)
```

Arguments

pgn_path	Path to a PGN file.

Value

A `data.frame` with one row per game and columns `Event`, `White`, `Black`, `Result`, `ECO`, `Plys`.

Examples

```
pgn_file <- system.file("extdata", "example.pgn", package = "ply")
if (file.exists(pgn_file)) {
  games <- ply_pgn_load_games(pgn_file)
  head(games)
}
```

---

ply_pgn_parse_tags *Parse PGN tag pairs from a raw game block*

Description

Parse PGN tag pairs from a raw game block

Usage

```
ply_pgn_parse_tags(game_text)
```

Arguments

game_text	A character string containing one PGN game block.

Value

A named list of tag key/value pairs.

Examples

```
g <- '[Event "Test"]\n[White "Alice"]\n[Black "Bob"]\n\n1.e4 e5 1-0'
tags <- ply_pgn_parse_tags(g)
tags$White  # "Alice"
```

ply_validate *Validate a position (piece counts, king count, etc.)*

Description

Validate a position (piece counts, king count, etc.)

Usage

```
ply_validate(state)
```

Arguments

state	A ChessState list.

Value

TRUE if the position is legal.

Examples

```
ply_validate(ply_game_init())  # TRUE
```

Index