Package ‘bracketeer’

February 25, 2026

Title Tournament Generator

Version 0.1.1

Description Create and manage tournament brackets for various competition formats including single elimination, double elimination, round robin, Swiss system, and group-stage-to-knockout tournaments. Provides tools for seeding, scheduling, recording results, and tracking standings.

URL `https://github.com/bbtheo/bracketeer`, `https://bbtheo.github.io/bracketeer/`

BugReports `https://github.com/bbtheo/bracketeer/issues`

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website pkgdown

NeedsCompilation no

Author Theo Blauberg [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0003-1780-0365`>)

Maintainer Theo Blauberg <`theo.blauberg@outlook.com`>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-02-25 19:20:07 UTC

Contents

add_stage 3
add_transition 3
advance 4

bottom_n 5
bottom_per_group 5
build 6
build_tournament 6
compute_tournament_rankings 7
double_elim 7
export_matches 8
export_standings 8
export_tournament_log 9
filter_by 9
from_previous 10
get_ready_stages 10
get_routing_log 11
group_stage_knockout 11
is_stage_complete 12
losers 12
matches 13
new_selector 13
previous_stage 14
print.group_stage_knockout 15
qualify_losers 15
qualify_remaining 16
rankings 16
remaining 17
result 17
results 18
round_robin 19
routing_log 19
set_outcome 20
single_elim 20
single_elim_stage 21
slice_per_group 23
slice_range 23
spec 24
split_stage 24
stage_status 25
standings 26
summary 26
swiss 27
teardown 28
top_n 28
top_per_group 29
tournament 29
tournament_spec 30
two_leg 30
validate 31
validate_tournament 32
validate_tournament_spec 32

winner 33
Index **34**

`add_stage` *Add a stage to a tournament specification*

Description

Add a stage to a tournament specification

Usage

```
add_stage(spec, stage_id, stage)
```

Arguments

`spec`	A `tournament_spec` object.
`stage_id`	Unique stage identifier.
`stage`	Stage definition object.

Value

Updated `tournament_spec`.

`add_transition` *Add a transition between stages*

Description

Add a transition between stages

Usage

```
add_transition(
  spec,
  from,
  to,
  rule = NULL,
  seeding = "by_source_rank",
  take = NULL,
  priority = 1L,
  consume = TRUE,
  allow_overlap = FALSE,
  transition_id = NULL
)
```

Arguments

`spec`	A `tournament_spec` object.
`from`	Source stage ID, or `from_previous()`.
`to`	Destination stage ID.
`rule`	Transition rule object (optional for MVP graph wiring).
`seeding`	Seeding policy label.
`take`	Selector object for transition participant selection.
`priority`	Transition resolution priority.
`consume`	Whether selected participants are consumed.
`allow_overlap`	Whether overlap is allowed across transitions.
`transition_id`	Transition ID. If `NULL`, deterministic auto-ID is used.

Value

Updated `tournament_spec`.

`advance` *Advance tournament to next round*

Description

Check if current round is complete and update bracket state.

Usage

```
advance(x, stage = NULL, ...)

## S3 method for class 'bracket'
advance(x, stage = NULL, ...)

## S3 method for class 'double_elim_bracket'
advance(x, stage = NULL, ...)

## S3 method for class 'group_stage_knockout'
advance(x, stage = NULL, ...)

## S3 method for class 'single_elim_bracket'
advance(x, stage = NULL, ...)

## S3 method for class 'swiss_bracket'
advance(x, stage = NULL, ...)

## S3 method for class 'tournament'
advance(x, stage = NULL, ...)
```

Arguments

`x`	A bracket object.
`stage`	Optional stage identifier for tournament methods.
`...`	Additional method-specific arguments.

Value

Updated bracket object

`bottom_n` *Select bottom ranked participants from source standings*

Description

Select bottom ranked participants from source standings

Usage

```
bottom_n(n)
```

Arguments

`n`	Positive integer count.

Value

A `bracketeer_selector` object.

`bottom_per_group` *Select bottom ranked participants per group*

Description

Select bottom ranked participants per group

Usage

```
bottom_per_group(n)
```

Arguments

`n`	Positive integer count per group.

Value

A `bracketeer_selector` object.

`build` *Build a live tournament runtime from a specification*

Description

Build a live tournament runtime from a specification

Usage

```
build(x, participants)
```

Arguments

`x`	A `bracketeer_spec` or `tournament_spec` object.
`participants`	Character vector of participant names, or a data.frame with a `name` column.

Value

A tournament runtime object.

Examples

```
my_spec <- spec() |>
  swiss("open", rounds = 3) |>
  single_elim("playoffs", take = top_n(4))

# Materialize with participants
trn <- build(my_spec, paste("Team", LETTERS[1:8]))
```

`build_tournament` *Build a tournament runtime from a tournament specification*

Description

Build a tournament runtime from a tournament specification

Usage

```
build_tournament(spec, participants)
```

Arguments

`spec`	A `tournament_spec` object.
`participants`	Character vector of participant names, or a data.frame with a `name` column.

Value

A `tournament` runtime object.

`compute_tournament_rankings` *Compute tournament rankings*

Description

Compute tournament rankings

Usage

```
compute_tournament_rankings(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame with `rank` and `participant`, or `NULL` when unavailable.

`double_elim` *Create a double elimination bracket*

Description

Double elimination tournament with winners and losers brackets.

Usage

```
double_elim(participants, ...)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a 'name' column and optional 'seed' column.
`...`	Additional arguments passed to bracket constructors or tournament stage-verb dispatch methods.

Value

A double_elim_bracket object

Examples

```
# Double elimination bracket (two losses to be eliminated)
trn <- tournament(paste("Team", LETTERS[1:8])) |>
  double_elim("bracket")

# After Swiss rounds
trn <- tournament(paste("Team", LETTERS[1:16])) |>
  swiss("open", rounds = 4) |>
  double_elim("playoffs", take = top_n(8))
```

`export_matches` *Export tournament matches across materialized stages*

Description

Export tournament matches across materialized stages

Usage

```
export_matches(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame with stage-tagged matches and compound match IDs.

`export_standings` *Export tournament standings across materialized stages*

Description

Export tournament standings across materialized stages

Usage

```
export_standings(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame with stage-tagged standings.

`export_tournament_log` *Export tournament routing log entries*

Description

Export tournament routing log entries

Usage

```
export_tournament_log(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame with one row per routing log entry.

`filter_by` *Select participants using a custom predicate function*

Description

Select participants using a custom predicate function

Usage

```
filter_by(fn)
```

Arguments

`fn`	A transition predicate function.

Value

A `bracketeer_selector` object.

`from_previous` *Resolve source stage from most recently defined stage order*

Description

Resolve source stage from most recently defined stage order

Usage

```
from_previous()
```

Value

Sentinel object to be resolved by `add_transition()`.

`get_ready_stages` *Get stage IDs currently ready to advance*

Description

Get stage IDs currently ready to advance

Usage

```
get_ready_stages(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Character vector of stage IDs in deterministic order.

`get_routing_log` *Get transition routing log entries*

Description

Get transition routing log entries

Usage

```
get_routing_log(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

List of routing log entries in append order.

`group_stage_knockout` *Create a group stage followed by knockout bracket*

Description

Create a group stage followed by knockout bracket

Usage

```
group_stage_knockout(participants, ...)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a 'name' column and optional 'seed' column.
`...`	Additional arguments passed to bracket constructors or tournament stage-verb dispatch methods.

Value

A group_stage_knockout object

`is_stage_complete` *Check whether a stage is complete*

Description

Check whether a stage is complete

Usage

```
is_stage_complete(x, ...)
```

Arguments

`x`	A stage bracket object or `tournament`.
`...`	Additional method-specific arguments.

Value

Logical scalar.

`losers` *Select losers from a source stage by elimination round*

Description

Select losers from a source stage by elimination round

Usage

```
losers(round = "all", stage = NULL, ordering = "elimination_round")
```

Arguments

`round`	One of `"all"`, `"latest"`, or an integer vector of rounds.
`stage`	Optional stage selector (reserved for future use).
`ordering`	Ordering mode: `"elimination_round"`, `"source_seed"`, or `"as_recorded"`.

Value

A `bracketeer_selector` object.

matches — *Inspect tournament matches*

Description

Inspect tournament matches

Usage

```
matches(x, stage = NULL, status = "pending")

## S3 method for class 'tournament'
matches(x, stage = NULL, status = "pending")
```

Arguments

`x`	A `tournament` object.
`stage`	Optional stage identifier.
`status`	One of `"pending"`, `"complete"`, or `"all"`.

Value

Data frame of matches.

Examples

```
trn <- tournament(c("A", "B", "C", "D")) |>
  round_robin("groups")

# Get pending matches
matches(trn, "groups")

# Get all matches across stages
matches(trn, status = "all")
```

new_selector — *Construct a selector object for transition* `take` *= routing*

Description

Construct a selector object for transition `take` = routing

Usage

```
new_selector(kind, params = list(), evaluator)
```

Arguments

`kind`	Selector kind label.
`params`	Selector parameters list.
`evaluator`	Function implementing selection logic.

Value

A `bracketeer_selector` object.

`previous_stage` *Resolve source stage from the immediately preceding stage*

Description

Alias for `from_previous()` used by the rewritten stage-verb API.

Usage

```
previous_stage()
```

Value

Sentinel object to be resolved by transition wiring.

Examples

```
teams <- paste("Team", LETTERS[1:8])

# Implicit: defaults to previous_stage()
trn <- tournament(teams) |>
  swiss("open", rounds = 3) |>
  single_elim("playoffs", take = top_n(4))

# Explicit: useful for branching
trn <- tournament(teams) |>
  round_robin("groups") |>
  single_elim("finals", from = previous_stage(), take = top_n(2))
```

print.group_stage_knockout — *Print bracketeer objects*

Description

Print bracketeer objects

Usage

```
## S3 method for class 'group_stage_knockout'
print(x, ...)

## S3 method for class 'bracket_match'
print(x, ...)

## S3 method for class 'bracket'
print(x, ...)

## S3 method for class 'double_elim_bracket'
print(x, ...)

## S3 method for class 'tournament'
print(x, ...)
```

Arguments

`x`	A bracket or match object.
`...`	Additional arguments (unused).

Value

The object, invisibly.

qualify_losers — *Select losers from a source stage by elimination round*

Description

Returns a transition rule function intended for use with `add_transition()`.

Usage

```
qualify_losers(round = "all", stage = NULL, ordering = "elimination_round")
```

Arguments

`round`	One of `"all"`, `"latest"`, or an integer vector of elimination rounds to include.
`stage`	Optional stage selector (reserved for future use).
`ordering`	Ordering mode: `"elimination_round"`, `"source_seed"`, or `"as_recorded"`.

Value

A transition rule function.

`qualify_remaining` *Select all entrants remaining in the transition source pool*

Description

Returns a transition rule function intended for use with `add_transition()`. During `advance()`, this selects all participants still available from the source stage after higher-priority consuming transitions have resolved.

Usage

```
qualify_remaining()
```

Value

A transition rule function.

`rankings` *Get tournament rankings*

Description

Get tournament rankings

Usage

```
rankings(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame of rankings.

`remaining` *Select entrants remaining in the current transition source pool*

Description

Select entrants remaining in the current transition source pool

Usage

```
remaining()
```

Value

A `bracketeer_selector` object.

`result` *Fluent tournament result entry helper*

Description

Convenience wrapper around `set_result()` for tournament workflows.

Usage

```
result(tournament, stage, match, score, overwrite = FALSE, auto_advance = NULL)
```

Arguments

`tournament`	A `tournament` object.
`stage`	Stage identifier containing the match.
`match`	Match identifier inside `stage`.
`score`	Numeric vector score payload. For a single match, pass `c(score1, score2)`.
`overwrite`	Logical; forwards to `set_result(..., overwrite = ...)`.
`auto_advance`	Optional logical override. If `NULL`, defaults to the tournament's `auto_advance` setting when present.

Value

Updated `tournament` object.

Examples

```
teams <- c("A", "B", "C", "D")
trn <- tournament(teams) |>
  round_robin("groups")

# Enter a single result
trn <- result(trn, "groups", match = 1, score = c(2, 1))
```

---

results *Fluent tournament batch result entry helper*

Description

Convenience wrapper for entering multiple match results for one stage.

Usage

```
results(tournament, stage, df, overwrite = FALSE, auto_advance = NULL)
```

Arguments

`tournament`	A `tournament` object.
`stage`	Stage identifier containing the matches.
`df`	Data frame with required columns: `match`, `score1`, `score2`.
`overwrite`	Logical; forwards to `result(..., overwrite = ...)`.
`auto_advance`	Optional logical override for the final row. If `NULL`, defaults to the tournament's `auto_advance` setting when present.

Value

Updated `tournament` object.

Examples

```
teams <- c("A", "B", "C", "D")
trn <- tournament(teams) |>
  round_robin("groups")

m <- matches(trn, "groups")
trn <- results(trn, "groups", data.frame(
  match  = m$match_id,
  score1 = c(2, 1, 3),
  score2 = c(1, 2, 0)
))
```

round_robin *Create a round robin tournament*

Description

Round robin tournament where each participant plays every other participant.

Usage

```
round_robin(participants, ...)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a 'name' column and optional 'seed' column.
`...`	Additional arguments passed to bracket constructors or tournament stage-verb dispatch methods.

Value

A round_robin_bracket object

Examples

```
# Simple round robin
trn <- tournament(c("A", "B", "C", "D")) |>
  round_robin("groups")

# Multiple groups (World Cup style)
teams <- paste("Team", sprintf("%02d", 1:32))
trn <- tournament(teams) |>
  round_robin("groups", groups = 8)
```

routing_log *Get transition routing log*

Description

Get transition routing log

Usage

```
routing_log(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame audit trail.

`set_outcome` *Configure tournament outcome depth*

Description

Configure tournament outcome depth

Usage

```
set_outcome(spec, track_placements = 1L)
```

Arguments

`spec`	A `tournament_spec` object.
`track_placements`	Number of placements to track.

Value

Updated `tournament_spec`.

`single_elim` *Create a single elimination bracket*

Description

Single elimination (knockout) tournament where losing a match eliminates the participant from the tournament.

Usage

```
single_elim(participants, ...)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a 'name' column and optional 'seed' column.
`...`	Additional arguments passed to bracket constructors or tournament stage-verb dispatch methods.

Value

A single_elim_bracket object

Examples

```
# Simple knockout bracket
trn <- tournament(paste("Team", LETTERS[1:8])) |>
  single_elim("bracket")

# Chain after group stage
trn <- tournament(c("A", "B", "C", "D")) |>
  round_robin("groups") |>
  single_elim("finals", take = top_n(2))
```

`single_elim_stage` *Create a stage specification*

Description

Stage specifications describe how to materialize a stage bracket from a participant set inside a `tournament_spec` graph.

Usage

```
single_elim_stage(
  seed = TRUE,
  third_place = FALSE,
  best_of = NULL,
  reseed = FALSE
)

double_elim_stage(
  seed = TRUE,
  grand_final_reset = TRUE,
  best_of = NULL,
  reseed = FALSE
)

round_robin_stage(
  home_away = FALSE,
  n_rounds = NULL,
  best_of = NULL,
  tiebreakers = NULL,
  groups = NULL
)

swiss_stage(
  rounds = NULL,
  seed = TRUE,
  allow_ties = TRUE,
  bye_points = 1,
  best_of = NULL,
  tiebreakers = NULL
)

group_stage_knockout_stage(
  groups = 2,
  advance_per_group = 2,
  seed = TRUE,
  group_home_away = FALSE,
  group_best_of = NULL,
  group_tiebreakers = NULL,
  knockout_type = "single_elim",
  knockout_seed = TRUE,
  third_place = FALSE,
  grand_final_reset = TRUE,
  knockout_best_of = NULL
)

two_leg_stage(
  seed = TRUE,
  third_place = FALSE,
  away_goals = TRUE,
  reseed = FALSE
)
```

Arguments

`seed`	Logical or character seed method.
`third_place`	Logical; include third-place match.
`best_of`	Optional best-of value (must be odd).
`reseed`	Logical; reseed between rounds for supported formats.
`grand_final_reset`	Logical; allow grand-final reset.
`home_away`	Logical; whether repeated pairings alternate home/away.
`n_rounds`	Number of round-robin cycles.
`tiebreakers`	Ordered tiebreakers.
`groups`	Number of groups.
`rounds`	Number of Swiss rounds.
`allow_ties`	Logical; whether ties are allowed.
`bye_points`	Points awarded for a bye.
`advance_per_group`	Number of qualifiers per group.
`group_home_away`	Logical; home/away behavior in groups.
`group_best_of`	Optional best-of in groups.

`group_tiebreakers`	Ordered group-stage tiebreakers.
`knockout_type`	Knockout format: `"single_elim"` or `"double_elim"`.
`knockout_seed`	Logical or character seed method for knockout.
`knockout_best_of`	Optional knockout best-of value.
`away_goals`	Logical; enable away-goals tiebreaker.

Value

A `stage_spec` object.

`slice_per_group` *Select an inclusive standings slice per group*

Description

Select an inclusive standings slice per group

Usage

```
slice_per_group(from, to)
```

Arguments

`from`	Positive integer starting position.
`to`	Positive integer ending position (must be >= `from`).

Value

A `bracketeer_selector` object.

`slice_range` *Select an inclusive standings slice*

Description

Select an inclusive standings slice

Usage

```
slice_range(from, to)
```

Arguments

`from`	Positive integer starting position.
`to`	Positive integer ending position (must be `>= from`).

Value

A `bracketeer_selector` object.

`spec` *Create a bracketeer tournament specification*

Description

Create a bracketeer tournament specification

Usage

```
spec()
```

Value

A `bracketeer_spec` object.

Examples

```
# Create a reusable tournament blueprint
my_spec <- spec() |>
  round_robin("groups") |>
  single_elim("finals", take = top_n(2))

# Build with different participant lists
trn1 <- build(my_spec, c("A", "B", "C", "D"))
trn2 <- build(my_spec, c("W", "X", "Y", "Z"))
```

`split_stage` *Add multiple transitions from one source stage*

Description

Convenience sugar for branching stage fan-out. Compiles into deterministic `add_transition()` calls.

Usage

```
split_stage(
  spec,
  from,
  into,
  priority_start = 1L,
  consume = TRUE,
  allow_overlap = FALSE,
  seeding = "by_source_rank"
)
```

Arguments

`spec`	A `tournament_spec` object.
`from`	Source stage ID, or `from_previous()`.
`into`	Named list mapping destination stage IDs to transition rules, or branch configs with a required `rule` field.
`priority_start`	Starting priority for branch transitions when a branch does not explicitly provide `priority`.
`consume`	Default `consume` value for branches.
`allow_overlap`	Default `allow_overlap` value for branches.
`seeding`	Default seeding policy for branches.

Value

Updated `tournament_spec`.

`stage_status` *Inspect tournament stage status*

Description

Inspect tournament stage status

Usage

```
stage_status(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Data frame with one row per stage.

standings — *Inspect tournament standings*

Description

Inspect tournament standings

Usage

```
standings(x, stage = NULL)

## S3 method for class 'tournament'
standings(x, stage = NULL)
```

Arguments

x	A `tournament` object.
stage	Optional stage identifier.

Value

Data frame of standings.

Examples

```
trn <- tournament(c("A", "B", "C", "D")) |>
  round_robin("groups")

# Enter some results
m <- matches(trn, "groups")
trn <- result(trn, "groups", m$match_id[1], score = c(2, 1))

# View current standings
standings(trn, "groups")
```

summary — *Summarize bracketeer objects*

Description

Summarize bracketeer objects

Usage

```
## S3 method for class 'bracket'
summary(object, ...)
```

Arguments

`object`	A bracket object.
`...`	Additional arguments (unused).

Value

The object, invisibly.

`swiss` *Create a Swiss-system tournament*

Description

Swiss system pairs participants by similar records each round.

Usage

```
swiss(participants, ...)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a 'name' column and optional 'seed' column.
`...`	Additional arguments passed to bracket constructors or tournament stage-verb dispatch methods.

Value

A swiss_bracket object

Examples

```
# Swiss system followed by top-cut playoffs
teams <- paste("Team", LETTERS[1:16])
trn <- tournament(teams) |>
  swiss("open", rounds = 5) |>
  single_elim("playoffs", take = top_n(8))
```

`teardown` *Teardown tournament state*

Description

For tournament runtimes, this un-materializes a stage and its downstream dependents so upstream results can be corrected and replayed.

Usage

```
teardown(x, stage = NULL, ...)

## S3 method for class 'bracket'
teardown(x, stage = NULL, ...)

## S3 method for class 'tournament'
teardown(x, stage = NULL, ...)
```

Arguments

`x`	A bracket or tournament object.
`stage`	Stage identifier to teardown (tournament method).
`...`	Additional method-specific arguments.

Value

Updated object.

`top_n` *Select top ranked participants from source standings*

Description

Select top ranked participants from source standings

Usage

```
top_n(n)
```

Arguments

`n`	Positive integer count.

Value

A `bracketeer_selector` object.

Examples

```
# Route top 4 to playoffs
trn <- tournament(paste("Team", LETTERS[1:8])) |>
  swiss("open", rounds = 3) |>
  single_elim("playoffs", take = top_n(4))
```

`top_per_group` *Select top ranked participants per group*

Description

Select top ranked participants per group

Usage

```
top_per_group(n)
```

Arguments

`n`	Positive integer count per group.

Value

A `bracketeer_selector` object.

Examples

```
# World Cup style: 8 groups, top 2 per group advance
teams <- paste("Team", sprintf("%02d", 1:32))
trn <- tournament(teams) |>
  round_robin("groups", groups = 8) |>
  single_elim("knockout", take = top_per_group(2))
```

`tournament` *Create an empty live tournament pipeline*

Description

Create an empty live tournament pipeline

Usage

```
tournament(participants, auto_advance = TRUE)
```

Arguments

`participants`	Character vector of participant names, or a data.frame with a `name` column.
`auto_advance`	Logical scalar. Stored as the runtime default for future result-entry helpers.

Value

A tournament runtime object with no stages materialized yet.

Examples

```
# Simple tournament with auto-advance
teams <- c("Lions", "Bears", "Eagles", "Wolves")
trn <- tournament(teams) |>
  round_robin("groups") |>
  single_elim("finals", take = top_n(2))

# Manual advance mode
trn_manual <- tournament(teams, auto_advance = FALSE) |>
  swiss("open", rounds = 3)
```

`tournament_spec` *Create a tournament specification graph*

Description

Construct a multi-stage tournament specification object.

Usage

```
tournament_spec()
```

Value

A `tournament_spec` object.

`two_leg` *Create a two-leg stage or bracket*

Description

Alias for `two_leg_knockout()` used by the tournament stage-verb API.

Usage

```
two_leg(participants, ...)
```

Arguments

`participants`	Participants, a spec object, or a tournament object.
`...`	Additional arguments forwarded to `two_leg_knockout()`.

Value

A bracket, spec, or tournament depending on `participants`.

Examples

```
# Two-leg knockout (Champions League style)
teams <- paste("Club", sprintf("%02d", 1:16))
trn <- tournament(teams) |>
  round_robin("groups", groups = 4) |>
  two_leg("knockouts", take = top_per_group(2))
```

`validate` *Validate a tournament spec preflight*

Description

Validate a tournament spec preflight

Usage

```
validate(x, n)
```

Arguments

`x`	A `bracketeer_spec` or `tournament_spec` object.
`n`	Participant count for feasibility checks.

Value

A preflight validation summary.

Examples

```
my_spec <- spec() |>
  round_robin("groups", groups = 4) |>
  single_elim("knockout", take = top_per_group(2))

# Check if 16 participants can work
validate(my_spec, n = 16)
```

validate_tournament *Dry-run preflight validation for tournament flow feasibility*

Description

Validates a `tournament_spec` against a participant count without running a live tournament. This preflight catches infeasible routing paths and stage size mismatches early.

Usage

```
validate_tournament(spec, n_participants)
```

Arguments

`spec`	A `tournament_spec` object.
`n_participants`	Positive integer participant count.

Value

A `tournament_validation` summary list.

validate_tournament_spec *Validate a tournament specification*

Description

Validate a tournament specification

Usage

```
validate_tournament_spec(spec)
```

Arguments

`spec`	A `tournament_spec` object.

Value

The validated `tournament_spec`.

winner — *Get tournament winner*

Description

Get tournament winner

Usage

```
winner(tournament)
```

Arguments

`tournament`	A `tournament` object.

Value

Winner name or `NA_character_`.

Examples

```
teams <- c("A", "B", "C", "D")
trn <- tournament(teams) |>
  round_robin("groups") |>
  single_elim("finals", take = top_n(2))

# ... enter all results ...

# Get the champion
winner(trn)
```

Index

add_stage, 3
add_transition, 3
advance, 4

bottom_n, 5
bottom_per_group, 5
build, 6
build_tournament, 6

compute_tournament_rankings, 7

double_elim, 7
double_elim_stage (single_elim_stage), 21

export_matches, 8
export_standings, 8
export_tournament_log, 9

filter_by, 9
from_previous, 10

get_ready_stages, 10
get_routing_log, 11
group_stage_knockout, 11
group_stage_knockout_stage (single_elim_stage), 21

is_stage_complete, 12

losers, 12

matches, 13

new_selector, 13

previous_stage, 14
print (print.group_stage_knockout), 15
print.group_stage_knockout, 15

qualify_losers, 15
qualify_remaining, 16

rankings, 16
remaining, 17
result, 17
results, 18
round_robin, 19
round_robin_stage (single_elim_stage), 21
routing_log, 19

set_outcome, 20
single_elim, 20
single_elim_stage, 21
slice_per_group, 23
slice_range, 23
spec, 24
split_stage, 24
stage_status, 25
standings, 26
summary, 26
swiss, 27
swiss_stage (single_elim_stage), 21

teardown, 28
top_n, 28
top_per_group, 29
tournament, 29
tournament_spec, 30
two_leg, 30
two_leg_stage (single_elim_stage), 21

validate, 31
validate_tournament, 32
validate_tournament_spec, 32

winner, 33