Package: gtree (via r-universe)

September 4, 2024

Type Package

Title gtree basic functionality to model and solve games

Version 0.1.1

Date 2021-03-01

Author Sebastian Kranz

Maintainer Sebastian Kranz <sebastian.kranz@uni-ulm.de>

Description gtree basic functionality to model and solve games

License GPL >= 2.0

URL https://github.com/skranz/gtree

Depends restorepoint,stringtools, yaml, data.table,dplyr, dplyrExtras, codeUtils, tidyr, digest, sys, lubridate, jsonlite, whisker, matrixStats, lazyeval

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.0.1

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

RemoteUrl https://github.com/skranz/gtree

RemoteRef master

RemoteSha dfcfa5b7a4f2101b11dfb7ed1d278a992657728e

Contents

action *Specify an action in a stage*

Description

Specify an action in a stage

Usage

```
action(name, set, strategyMethodDomain = NULL, ...)
```

Arguments

`name`	The variable name of the action
`set`	The set of different action values. Can be a formula that depends on other game variables.
`strategyMethodDomain`	if not NULL the action shall be specified via strategy method in an experiment. State the variable name upon which the action conditions Only used when running an experiment or analysing experimental data

See Also

Other Define Stage: `natureMove`, `stage`

`case_distinction` *A simple function to define case distinctions*

Description

A simple function to define case distinctions

Usage

```
case_distinction(...)
```

See Also

Other Helper Functions: `is_true`

`eq_cond_expected_outcomes` *Return conditional expected equilibrium outcomes*

Description

Return conditional expected equilibrium outcomes

Usage

```
eq_cond_expected_outcomes(game, ..., fixed.list = list(...),
  fixed.vars = NULL, like.factor = NULL)
```

Arguments

`game`	the game object for which equilibria were computed e.g. with `game_solve`.
`...`	variable names and their assumed value. We set the probabilities of the conditioned variable values to 1. These correspond to equilibrium outcomes given an unexpected tremble that makes the variables take the specified values. Variables can take multiple values. We then compute conditional equilibrium outcomes for each combination of values
`fixed.list`	Alternativly to ... a named list with values to fix.
`fixed.vars`	Alternative to ... or fixed.list, a vector of variable names. If provided, we compute the conditional expected outcomes holding fixed every possible combination of the variables stated in fixed.vars
`like.factor`	an optional character vector of names of numerical variables that shall be presented like qualitative variables.

See Also

Other eq: `eq_cond_outcomes`, `eq_expected_outcomes`, `eq_li`, `eq_outcomes`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`eq_cond_outcomes` *Return conditional equilibrium outcomes*

Description

Return conditional equilibrium outcomes

Usage

```
eq_cond_outcomes(game, ..., fixed.list = list(...), fixed.vars = NULL)
```

Arguments

`game`	the game object for which equilibria were computed e.g. with `game_solve`.
`...`	variable names and their assumed value. We set the probabilities of the conditioned variable values to 1. These correspond to equilibrium outcomes given an unexpected tremble that makes the variables take the specified values. Variables can take multiple values. We then compute conditional equilibrium outcomes for each combination of values
`fixed.list`	Alternativly to ... a named list with values to fix.
`fixed.vars`	Alternative to ... or fixed.list, a vector of variable names. If provided, we compute the conditional expected outcomes holding fixed every possible combination of the variables stated in fixed.vars

See Also

Other eq: `eq_cond_expected_outcomes`, `eq_expected_outcomes`, `eq_li`, `eq_outcomes`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`eq_expected_outcomes` *Return a data frame of expected equilibrium outcomes*

Description

Each row will describe a possible expected equilibrium outcome. For numerical variables like `payoff_1` the expected value on the equilibrium path is returned.

Usage

```
eq_expected_outcomes(game, like.factor = NULL)
```

Arguments

`game`	the game object for which prevoiously equilibria were computed e.g. with `game_solve`.
`like.factor`	an optional character vector of names of numerical variables that shall be presented like qualitative variables.

Details

For qualitative variables, we generate a string like `"accept(0.3),reject(0.2)"` describing the moves that occur with positive probability and those probabilities on the equilibrium path.

See Also

Other eq: `eq_cond_expected_outcomes`, `eq_cond_outcomes`, `eq_li`, `eq_outcomes`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`eq_li` *Return the computed equilibria using the internal representation*

Description

Return the computed equilibria using the internal representation

Usage

```
eq_li(game, ...)
```

See Also

Other eq: `eq_cond_expected_outcomes`, `eq_cond_outcomes`, `eq_expected_outcomes`, `eq_outcomes`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`eq_outcomes` *Return a data frame of all equilibrium outcomes*

Description

If we have mixed strategies or moves of nature an equilibrium outcome will consist of several rows. One row for each pure outcome that occurs with positive probability.

Usage

```
eq_outcomes(game, add.move.probs = FALSE)
```

Arguments

`game`	the game object for which previously equilibria were computed e.g. with `game_solve`.
`add.move.probs`	if `TRUE` add for each action (and move of nature) the probability that the actual value has been chosen in the corresponding information set (node).

Details

Typically `eq_expected_outcomes` will deliver a version that is easier to read. It will take expected values and reduce each outcome to one row.

Yet `eq_outcomes` may be more useful for automatical analysis.

See Also

Other eq: `eq_cond_expected_outcomes`, `eq_cond_outcomes`, `eq_expected_outcomes`, `eq_li`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

Other eq: `eq_cond_expected_outcomes`, `eq_cond_outcomes`, `eq_expected_outcomes`, `eq_li`, `eq_tables`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`eq_tables` *Return solved equilibrium in a table format*

Description

Best take a look at the Vignettes to understand this format.

Usage

```
eq_tables(game, reduce.tables = TRUE, combine = 2,
  eq.ind = seq_along(game$eq.li), ignore.keys = NULL,
  eq.li = game$eq.li[eq.ind], ...)
```

Arguments

`reduce.tables`	(default = TRUE). Shall we try to reduce the rows and columns of the key tables be reduced to get a subset of neccessary keys that perfectly predict the chosen value of an action?
`combine`	if 0 generate separate tables for each equilibrium. If 1 bind the tables of each variable over all equilibria. If 2 (default) also collapse the rows that are the same for different equilibria and add a column eq.inds that contains all equilibrium numbers as a comma separated string
`eq.ind`	Vector of integers specifying the indices of all equilibria that shall be considered. By default all equilibria.
`ignore.keys`	A character vector of variables that will always be removed from the key variables, without any check whether they are neccessary or not.

See Also

Other eq: `eq_cond_expected_outcomes`, `eq_cond_outcomes`, `eq_expected_outcomes`, `eq_li`, `eq_outcomes`, `game_gambit_solve_qre`, `game_gambit_solve`, `game_solve_spe`

`game_change_param` *Changes one or several parameters of a game*

Description

For an already compiled game, we try to change parameters in a fashion that is faster than a complete recompilation.

Usage

```
game_change_param(game, ..., params = list(),
  verbose = isTRUE(game$options$verbose > 0))
```

See Also

Other Modify Game: `game_copy`, `game_set_preferences`, `pref_change_params`

Other Game Parameters: `make_game_params`

`game_compile` *Compile a game defined with* `new_game`

Description

Compile a game defined with `new_game`

Usage

```
game_compile(game, branching.limit = 10000, for.internal.solver = FALSE,
  add.sg = for.internal.solver, add.spi = for.internal.solver,
  add.spo = for.internal.solver, force = FALSE,
  verbose = game$options$verbose, ...)
```

See Also

Other Build Game: `make_game_options`, `make_game_params`, `stage`

`game_copy` *Make a deep copy of a game*

Description

Make a deep copy of a game

Usage

```
game_copy(game)
```

See Also

Other Modify Game: `game_change_param`, `game_set_preferences`, `pref_change_params`

`game_fix_actions` *Fix move probabilities of actions*

Description

The function corresponds the provided actions into moves of nature with specified move probabilities. Can be a useful step when checking for existence of equilibria with particular structure.

Usage

```
game_fix_actions(game, ..., actions = list(...), tremble.prob = NULL)
```

Arguments

`...`	directly the named arguments from which `actions` will be constructed
`actions`	a named list. The names correspond to action names. The default value to fix mixed strategies is a table that specifies conditional move probabilities (see example). If you want to fix pure actions you can also provide arguments as in `game_fix_action_preferences`.
`tremble.prob`	If a positive number, we assume that with this probability the player trembles and then chooses a random action with uniform probability. Trembles can be useful to enforce some sequential rationality in continuation play, but note that uniform trembles are not neccessarily the correct form of trembles to find sequential equilibria or trembling hand perfect equilibria.

Details

For fixing pure strategies `game_fix_action_preferences` is preferable when using the `gambit-logit` solver that can find sequential equilibria, by using logit trembles.

See Also

Other Fix Actions: game_fix_action_preferences, game_prefer_outcomes

---

game_fix_action_preferences

Set add a large amount of utility if a player plays a particular action

---

Description

Allows to study the game under the assumption that a player strongly prefers to chose one particular move of an action variable.

Usage

```
game_fix_action_preferences(game, ..., actions = list(...), util.add = 1000)
```

Arguments

`...`	directly the named arguments from which `actions` will be constructed
`actions`	a named list. The names correspond to action names and the values either to fixed values of the action or to a formula. If it is a formula the action value can depend on earlier computed variables.

Details

If you want to fix mixed strategies, use the he related function game_fix_action_preferences transforms the corresponding action into a move of nature.

For fixing pure strategies game_fix_action_preferences is preferable when using the `gambit-logit` solver that can find sequential equilibria, by using logit trembles.

See Also

Other Fix Actions: game_fix_actions, game_prefer_outcomes

Other Preferences: game_prefer_outcomes, game_set_preferences, pref_change_params, pref_custom, pref_envy, pref_heterogeneous_players, pref_ineqAv, pref_lossAv, pref_payoff

game_gambit_solve *Solve equilibria of a game using Gambit*

Description

You need to install Gambit http://www.gambit-project.org to use this function.

Usage

```
game_gambit_solve(game, gambit.command = NULL, mixed = FALSE,
  just.spe = TRUE, qre.lambda = NULL,
  gambit.dir = first.non.null(getOption("gtree.gambit.dir"), ""),
  efg.dir = NULL, efg.file = NULL, verbose = isTRUE(game$options$verbose
  >= 1), add.q.flag = TRUE, ...)
```

Arguments

`game`	the game object created with new_game
`gambit.command`	A Gambit command line command with options but not file name. For example `"gambit-enummixed -q"` to compute all extreme point mixed equilibria. The different Gambit command line solvers are described here: http://www.gambit-project.org/gambit16/16.0.0/tools.html If left as NULL a default gambit command line solver with appropriate arguments will be chosen, depending on your arguments for mixed and just.spe
`mixed`	relevant if no explicit gambit.command is given. If FALSE (default) only pure strategy equilibria will be computed, otherwise try to compute one mixed equilibrium.
`just.spe`	if TRUE compute only SPE. If FALSE all NE will be computed.
`qre.lambda`	if not NULL compute a logit QRE equilibrium using the gambit-logit solver and the specified value of lambda.
`gambit.dir`	The directory where to find the Gambit command line solvers. Ideally, you put this directory into the search path of your system and can keep the default `gambit.dir = ""`. To globally change the default directory adapt the following code `options(gtree.gambit.dir = "/PATH/TO/GAMBIT")`
`efg.dir`	To solve via Gambit we first write the game tree into an .efg file. If `efg.dir` is NULL (default), the file will be written to a temporary directory. But you can also specify a custom directory here, e.g. if you want to take a look at the file.
`efg.file`	If NULL a default file name for the efg file will be generated based on the name of the game and the specified preferences. But you can specify a custom name here.
`verbose`	if TRUE show some extra information
`add.q.flag`	The gambit command line solver should always be called with the option "-q" for gtree to be able to parse the returned output. If add.q.flag is TRUE we will add this flag if you have not yet added it to your `gambit.command`

See Also

Other eq: eq_cond_expected_outcomes, eq_cond_outcomes, eq_expected_outcomes, eq_li, eq_outcomes, eq_tables, game_gambit_solve_qre, game_solve_spe

Other Gambit: game_write_efg

---

game_gambit_solve_qre *Solve for quantal response equilibria using Gambit*

Description

This function computes logit agent quantal response equilibria using the Gambit solver gambit-logit. For a short description see the Wikipedia article and the gambit-logit solver's https://gambitproject.readthedocs.io/en/latest/to logit-compute-quantal-response-equilbria. Details are in the article https://link.springer.com/article/10.1007/s00199-009-0443-3 by Theodore Turocy. But unfortunately, the article can only be found behind a pay wall.

Usage

```
game_gambit_solve_qre(game, gambit.command = "gambit-logit -q -l",
  gambit.dir = "", efg.file = NULL, efg.dir = NULL,
  verbose = isTRUE(game$options$verbose >= 1))
```

Details

For a description of the arguments see game_gambit_solve

See Also

Other eq: eq_cond_expected_outcomes, eq_cond_outcomes, eq_expected_outcomes, eq_li, eq_outcomes, eq_tables, game_gambit_solve, game_solve_spe

---

game_prefer_outcomes *Set add a large amount of utility if a player plays a particular action*

Description

Allows to study the game under the assumption that a player strongly prefers to chose one particular move of an action variable.

Usage

```
game_prefer_outcomes(game, player1 = NULL, player2 = NULL, player3 = NULL,
  ..., player.prefs = list(player1 = player1, player2 = player2, player3 =
  player3, ...))
```

Arguments

`player1`	A formula describing which utility levels should be added to the current utility function of player 1 (see example). If NULL we don't add utilities for player 1. Similar for the other players 2-4.
`...`	additional formulas for games with more than 4 players.
`player.prefs`	by default equal to `list(player1,player2,player3, player4,...)`. Can be manually provided.

Details

If you want to fix mixed strategies, use the he related function `game_fix_action_preferences` transforms the corresponding action into a move of nature.

For fixing pure strategies `game_fix_action_preferences` is preferable when using the `gambit-logit` solver that can find sequential equilibria, by using logit trembles.

See Also

Other Fix Actions: `game_fix_action_preferences`, `game_fix_actions`

Other Preferences: `game_fix_action_preferences`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

`game_set_options` *Change options of an already created game object*

Description

See `make_game_options` for a description of the available options.

Usage

```
game_set_options(game, ...)
```

See Also

Other Game Options: `make_game_options`

game_set_preferences *Set players' preferences*

Description

This function sets players preferences to a parametrized preference type. To specify completely custom preferences use game_set_util_fun instead.

Usage

```
game_set_preferences(game, pref)
```

Arguments

game	The game object
pref	A preference created with a function starting with `pref_`, like e.g. `pref_ineqAv(alpha=1, beta=0.5)`. Use `pref_custom` to specify custom preferences.

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

Other Modify Game: `game_change_param`, `game_copy`, `pref_change_params`

game_solve_spe *Solve equilibria of a game*

Description

With the default arguments the internal gtree solver is used to find all pure strategy subgame perfect equilibria of the game.

Usage

```
game_solve_spe(game, mixed = FALSE, just.spe = TRUE, use.gambit = mixed |
  !just.spe, verbose = isTRUE(game$options$verbose >= 1),
  gambit.command = NULL, ...)
```

Arguments

game	the game object created with new_game
use.gambit	solve via Gambit. Changing `mixed` or `just.spe` or specifying a `gambit.command` has only impact if `use.gambit=TRUE`. See `game_gambit_solve` for details.

See Also

Other eq: eq_cond_expected_outcomes, eq_cond_outcomes, eq_expected_outcomes, eq_li, eq_outcomes, eq_tables, game_gambit_solve_qre, game_gambit_solve

---

game_write_efg *Write game as a Gambit efg file*

Description

Write game as a Gambit efg file

Usage

```
game_write_efg(game, file.with.dir = file.path(dir, file),
  file = tg.efg.file.name(game$tg), dir = getwd(),
  verbose = !isTRUE(game$options$verbose == 0))
```

Arguments

game	The game object
file.with.dir	The file with full path. If NULL create a default name
file	The file name without directory
dir	The directory of a file

See Also

Other Gambit: game_gambit_solve

---

get_outcomes *Return a data frame of all possible outcomes*

Description

Return a data frame of all possible outcomes

Usage

```
get_outcomes(game, reduce.cols = TRUE)
```

Arguments

game	the game object defined with new_game and being compiled with game_compile or after a call of game_solve.
reduce.cols	if TRUE remove some technical columns

`is_true` *Returns logical vector replacing NA by FALSE*

Description

Returns logical vector replacing NA by FALSE

Usage

```
is_true(vec)
```

Arguments

`vec`	A vector of logical values (possible containing NA) or values that can be transformed to logicals

See Also

Other Helper Functions: `case_distinction`

`make_game_options` *Specify the game options inside* `new_game`

Description

Specify the game options inside `new_game`

Usage

```
make_game_options(verbose = TRUE, ...)
```

See Also

game_set_options

Other Build Game: `game_compile`, `make_game_params`, `stage`

Other Game Options: `game_set_options`

make_game_params *Specify the game parameters This function is only to be used inside* new_game. *To change the parameters of an existing game call* game_change_params.

Description

Specify the game parameters This function is only to be used inside new_game. To change the parameters of an existing game call game_change_params.

Usage

```
make_game_params(numPlayers = 2, ...)
```

See Also

Other Build Game: game_compile, make_game_options, stage

Other Game Parameters: game_change_param

natureMove *Specify a random move of nature in a stage*

Description

Specify a random move of nature in a stage

Usage

```
natureMove(name, set, probs = NULL, table = NULL, fixed = NULL,
  tremble.prob = NULL, ...)
```

Arguments

name	The variable name of the variable
set	The set of different values. Can be a rhs only formula.
probs	The probability of each element in set. If NULL all moves are equally likely. Can be a rhs formula

See Also

Other Define Stage: action, stage

`new_game` *Create a new gtree game*

Description

See the examples on gtree website for detailed explanation.

Usage

```
new_game(gameId, params = game_params(), options = make_game_options(),
  stages, variant = "", check = TRUE)
```

`pref_change_params` *Change the parameters of a preference object*

Description

Change the parameters of a preference object

Usage

```
pref_change_params(pref, ..., params = list(), label = NULL,
  players = 1:2, numPlayers = length(players))
```

See Also

Other Modify Game: `game_change_param`, `game_copy`, `game_set_preferences`

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

`pref_custom` *Create a custom preference*

Description

Create a custom preference

Usage

```
pref_custom(..., params = NULL, label = "custom")
```

Arguments

`...`	Unquoted that describe the utility as a function of the parameters of the game and possible preference parameters. Should be ordered by players. Names are irrelevant.
`params`	An optional list of parameters that are used in the formulas above
`label`	A label for the preference, should contain info about the parameters
`type`	A general type label independet of the parameters

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

`pref_envy` *Fehr-Schmidt inequality aversion with envy only*

Description

Fehr-Schmidt inequality aversion with envy only

Usage

```
pref_envy(alpha = 0, player = 1:numPlayers, numPlayers = 2, ...)
```

Arguments

`alpha`	the degree of envy
`player`	player(s) for which the preferences apply. Per default 1:2
`numPlayers`	number of players in game per default 2

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

pref_heterogeneous_players
Combine preferences for different players

Description

Combine preferences for different players

Usage

```
pref_heterogeneous_players(..., prefs = list(...), label = NULL)
```

Arguments

`...`	all preferences ordered by players
`prefs`	alternatively the preferences as a list object
`label`	optional label of preferences. If NULL the individual labels will be pasted together
`type`	label of the combined preference type

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_ineqAv`, `pref_lossAv`, `pref_payoff`

pref_ineqAv
Fehr-Schmidt inequality aversion.

Description

Fehr-Schmidt inequality aversion.

Usage

```
pref_ineqAv(alpha = 0, beta = 0, player = 1:numPlayers, numPlayers = 2,
  ...)
```

Arguments

`alpha`	the degree of envy
`beta`	the degree of guilt
`player`	player(s) for which the preferences apply. Per default 1:2
`numPlayers`	number of players in game per default 2

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_lossAv`, `pref_payoff`

`pref_lossAv` *'Linear loss aversion preferences with a single reference point*

Description

'Linear loss aversion preferences with a single reference point

Usage

```
pref_lossAv(lambda = 2, r = 0, player = 1:numPlayers, numPlayers = 2)
```

Arguments

`lambda`	factor by which losses loom larger than gains (default = 2)
`r`	The reference point, by default 0. Can be a vector in order to have different reference points for different players.
`player`	player(s) for which the preferences apply. Per default 1:2
`numPlayers`	number of players in game per default 2

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_payoff`

`pref_payoff` *Utility is equal to monetary payoff.*

Description

This means the player is simply a risk neutral expected payoff maximizer.

Usage

```
pref_payoff(player = 1:2, ...)
```

Arguments

`player`	player(s) for which the preferences apply. Per default 1:2

See Also

Other Preferences: `game_fix_action_preferences`, `game_prefer_outcomes`, `game_set_preferences`, `pref_change_params`, `pref_custom`, `pref_envy`, `pref_heterogeneous_players`, `pref_ineqAv`, `pref_lossAv`

`stage` *Specify a stage for a game*

Description

Specify a stage for a game

Usage

```
stage(name, player = NULL, condition = NULL, observe = NULL,
  compute = NULL, nature = NULL, actions = NULL, ...)
```

Arguments

`name`	Name of the stage
`player`	The player who acts in this stage. Can be a rhs formula. If an action is chosen in the stage, there must be a unique player. If it is a stage in which no actions take place, the player variable multiple players can be set. Each player observes the variables specified under observe.
`condition`	A logical condition specifying whether the stage will be run. Can be a rhs formula. If it evaluates to FALSE the stage will not be shown, i.e. no observations are made and no actions are chosen. Also no computations in this stage take place.
`observe`	A vector of variable names specifying which variables are observed by the player(s) at this stage. Is relevant to correctly specify the information sets in the extensive form game.
`compute`	A list of formulas like 'compute=list(payoff_1 ~ x-5)'. The lhs specifies a variable name and the rhs a DETERMINISTIC formula. The variables are computed at the beginning of the stage before actions and moves of nature take place. This means they can be used e.g. in formulas for action sets of the same stage.
`nature`	A list of moves of nature, i.e. random variables from a finite set. E.g. nature=list(natureMove("proposer",c(1,2),prob=c(0.4,0.6)).
`actions`	A list of actions. E.g. actions=list(action("offer",~0:cake_size)

See Also

Other Define Stage: `action`, `natureMove`

Other Build Game: `game_compile`, `make_game_options`, `make_game_params`

Index