

# Package ‘tictactoe’

May 8, 2026

**Type** Package

**Title** Tic-Tac-Toe Game

**Version** 0.2.2

**Description** Implements tic-tac-toe game to play on console, either with human or AI players.  
Various levels of AI players are trained through the Q-learning algorithm.

**License** MIT + file LICENSE

**LazyData** TRUE

**RoxygenNote** 6.0.1

**Depends** R (>= 2.10)

**Imports** hash, stats

**Suggests** testthat, combiter, dplyr, tidyr, reshape2, ggplot2

**URL** <https://github.com/kota7/tictactoe>

**BugReports** <https://github.com/kota7/tictactoe/issues>

**NeedsCompilation** no

**Author** Kota Mori [aut, cre]

**Maintainer** Kota Mori <kmori05@gmail.com>

**Repository** CRAN

**Date/Publication** 2017-05-26 15:33:31 UTC

## Contents

equivalent_states . . . . .	2
hash-ops . . . . .	2
ttt . . . . .	3
ttt_ai . . . . .	4
ttt_game . . . . .	5
ttt_human . . . . .	5
ttt_qlearn . . . . .	6
ttt_simulate . . . . .	7
vectorized-hash-ops . . . . .	8
xhash . . . . .	9

**Index****10**


---

equivalent_states	<i>Equivalent States</i>
-------------------	--------------------------

---

**Description**

Returns a set of equivalent states and actions

**Usage**

```
equivalent_states(state)
```

```
equivalent_states_actions(state, action)
```

**Arguments**

state            state, 3x3 matrix

action           integer vector of indices (1 to 9)

**Value**

equivalent\_states returns a list of state matrices

equivalent\_states\_actions returns a list of two lists: states, the set of equivalent states and actions, the set of equivalent actions

---

hash-ops	<i>Hash Operations for Single State</i>
----------	---

---

**Description**

Hash Operations for Single State

**Usage**

```
haskey(x, ...)
```

```
## S3 method for class 'xhash'
x[state, ...]
```

```
## S3 replacement method for class 'xhash'
x[state, ...] <- value
```

```
## S3 method for class 'xhash'
haskey(x, state, ...)
```

**Arguments**

x	object
...	additional arguments to determine the key
state	state object
value	value to assign

**Value**

- haskey returns a logical
- `[` returns a reference to the object
- `[<-` returns a value

---

ttd

*Play Tic-Tac-Toe Game*


---

**Description**

Start tic-tac-toe game on the console.

**Usage**

```
ttd(player1 = ttd_human(), player2 = ttd_human(), sleep = 0.5)
```

**Arguments**

player1, player2	objects that inherit ttd_player class
sleep	interval to take before an AI player to make decision, in second

**Details**

At default, the game is played between humans. Set player1 or player2 to ttd\_ai() to play against an AI player. The strength of the AI can be adjusted by passing the level argument (0 (weakest) to 5 (strongest)) to the ttd\_ai function.

To input your move, type the position like "a1". Only two-length string consisting of an alphabet and a digit is accepted. Type "exit" to finish the game.

You may set both player1 and player2 as AI players. In this case, the game transition is displayed on the console without human inputs. For conducting a large sized simulations of games between AIs, refer to [ttd\\_simulate](#)

**See Also**

[ttd\\_ai](#), [ttd\\_human](#), [ttd\\_simulate](#)

## Examples

```
## Not run:  
ttt(ttt_human(), ttt_random())  
  
## End(Not run)
```

---

ttt_ai	<i>Tic-Tac-Toe AI Player</i>
--------	------------------------------

---

## Description

Create an AI tic-tac-toe game player

## Usage

```
ttt_ai(name = "ttt AI", level = 0L)  
  
ttt_random(name = "random AI")
```

## Arguments

name	player name
level	AI strength. must be Integer 0 (weakest) to 5 (strongest)

## Details

level argument controls the strength of AI, from 0 (weakest) to 5 (strongest). `ttt_random` is an alias of `ttt_ai(level = 0)`.

A `ttt_ai` object has the `getmove` function, which takes `ttt_game` object and returns a move considered as optimal. `getmove` function is designed to take a `ttt_game` object and returns a move using the policy function.

The object has the `value` and `policy` functions. The `value` function maps a game state to the evaluation from the first player's viewpoint. The `policy` function maps a game state to a set of optimal moves in light of the value evaluation. The functions have been trained through the Q-learning.

## Value

`ttt_ai` object

---

ttt_game	<i>Tic-Tac-Toe Game</i>
----------	-------------------------

---

**Description**

Object that encapsulates a tic-tac-toe game.

**Usage**

```
ttt_game()
```

**Value**

ttt\_game object

**Examples**

```
x <- ttt_game()
x$play(3)
x$play(5)
x$show_board()
```

---

ttt_human	<i>Human Tic-Tac-Toe Player</i>
-----------	---------------------------------

---

**Description**

Create an human tic-tac-toe player

**Usage**

```
ttt_human(name = "no name")
```

**Arguments**

name            player name

**Value**

ttt\_human object

ttt\_qlearn

*Q-Learning for Training Tic-Tac-Toe AI***Description**

Train a tic-tac-toe AI through Q-learning

**Usage**

```
ttt_qlearn(player, N = 1000L, epsilon = 0.1, alpha = 0.8, gamma = 0.99,
           simulate = TRUE, sim_every = 250L, N_sim = 1000L, verbose = TRUE)
```

**Arguments**

player	AI player to train
N	number of episode, i.e. training games
epsilon	fraction of random exploration move
alpha	learning rate
gamma	discount factor
simulate	if true, conduct simulation during training
sim_every	conduct simulation after this many training games
N_sim	number of simulation games
verbose	if true, progress report is shown

**Details**

This function implements Q-learning to train a tic-tac-toe AI player. It is designed to train one AI player, which plays against itself to update its value and policy functions.

The employed algorithm is Q-learning with epsilon greedy. For each state  $s$ , the player updates its value evaluation by

$$V(s) = (1 - \alpha)V(s) + \alpha\gamma\max_s V(s')$$

if it is the first player's turn. If it is the other player's turn, replace *max* by *min*. Note that  $s'$  spans all possible states you can reach from  $s$ . The policy function is also updated analogously, that is, the set of actions to reach  $s'$  that maximizes  $V(s')$ . The parameter  $\alpha$  controls the learning rate, and *gamma* is the discount factor (earlier win is better than later).

Then the player chooses the next action by  $\epsilon$ -greedy method; Follow its policy with probability  $1 - \epsilon$ , and choose random action with probability  $\epsilon$ .  $\epsilon$  controls the ratio of explorative moves.

At the end of a game, the player sets the value of the final state either to 100 (if the first player wins), -100 (if the second player wins), or 0 (if draw).

This learning process is repeated for N training games. When *simulate* is set true, simulation is conducted after *sim\_every* training games. This would be useful for observing the progress of training. In general, as the AI gets smarter, the game tends to result in draw more.

See Sutton and Barto (1998) for more about the Q-learning.

**Value**

data.frame of simulation outcomes, if any

**References**

Sutton, Richard S and Barto, Andrew G. Reinforcement Learning: An Introduction. The MIT Press (1998)

**Examples**

```
p <- ttt_ai()
o <- ttt_qlearn(p, N = 200)
```

---

ttt_simulate	<i>Simulate Tic-Tac-Toe Games between AIs</i>
--------------	---

---

**Description**

Simulate Tic-Tac-Toe Games between AIs

**Usage**

```
ttt_simulate(player1, player2 = player1, N = 1000L, verbose = TRUE,
             showboard = FALSE, pauseif = integer(0))
```

**Arguments**

player1, player2	AI players to simulate
N	number of simulation games
verbose	if true, show progress report
showboard	if true, game transition is displayed
pauseif	pause the simulation when specified results occur. This can be useful for explorative purposes.

**Value**

integer vector of simulation outcomes

**Examples**

```
res <- ttt_simulate(ttt_ai(), ttt_ai())
prop.table(table(res))
```

---

vectorized-hash-ops    *Vectorized Hash Operations*

---

## Description

Vectorized Hash Operations

## Usage

```
haskeys(x, ...)  
  
setvalues(x, ...)  
  
getvalues(x, ...)  
  
## S3 method for class 'xhash'  
getvalues(x, states, ...)  
  
## S3 method for class 'xhash'  
setvalues(x, states, values, ...)  
  
## S3 method for class 'xhash'  
haskeys(x, states, ...)
```

## Arguments

x	object
...	additional arguments to determine the keys
states	state object
values	values to assign

## Value

- haskeys returns a logical vector
- setvalues returns a reference to the object
- getvalues returns a list of values

---

`xhash`*Create Hash Table for Generic Keys*

---

**Description**

Create Hash Table for Generic Keys

**Usage**

```
xhash(convfunc = function(state, ...) state, convfunc_vec = function(states,
...) unlist(Map(convfunc, states, ...)), default_value = NULL)
```

**Arguments**

<code>convfunc</code>	function that converts a game state to a key. It must take a positional argument <code>state</code> and keyword arguments represented by <code>...</code> , and returns a character.
<code>convfunc_vec</code>	function for vectorized conversion from states to keys. This function must receive a positional argument <code>states</code> and keyword arguments <code>...</code> and returns character vector. By default, it tries to vectorize <code>convfunc</code> using <code>Map</code> . User may specify more efficient function if any.
<code>default_value</code>	value to be returned when a state is not recorded in the table.

**Value**

xhash object

# Index

[.xhash (hash-ops), 2  
[<-.xhash (hash-ops), 2

equivalent\_states, 2  
equivalent\_states\_actions  
    (equivalent\_states), 2

getvalues (vectorized-hash-ops), 8

hash-ops, 2  
haskey (hash-ops), 2  
haskeys (vectorized-hash-ops), 8

setvalues (vectorized-hash-ops), 8

ttt, 3  
ttt\_ai, 3, 4  
ttt\_game, 5  
ttt\_human, 3, 5  
ttt\_qlearn, 6  
ttt\_random (ttt\_ai), 4  
ttt\_simulate, 3, 7

vectorized-hash-ops, 8

xhash, 9