## Arena & Arena Train Elements

### Agent Type

* **T****DS**: TD Learning agent according to [Sutton&Bonde 1993] with a linear net or backprop net as approximator for the value function.
* **TD-****Ntuple**: TD-Learning agent using N-tuples as features, with a linear net (no hidden layer) as approximator for the value function
* **MC**: Monte Carlo
* **MC****TS**: Monte Carlo Tree Search
* **MCTS Expectimax**: Monte Carlo Tree Search for nondeterministic games
* **Mi****nimax**: game-tree search agent using Minimax strategy (Alpha-Beta-search). Realizes perfect play for TicTacToe, but may ‘explode’ for more complex games.
* **Ra****ndom**: random playing agent
* **Hu****man**: Human player [[1]](#footnote-1)

### Play

Play a game with the agents currently selected as “X” or “O” in the [Agent Type](#AgentType) combo boxes. If the currently selected agent (“X” or “O”) is not yet trained or the trained agent differs from the one in the combo boxes, an error message is displayed. If one of the selected agents is [Human](#Human), the user has to fill in the appropriate moves.

The starting player is always “X” (2-player games).

### InspectV

Inspect the value function V(**s’**) of the X-Player. The X-Player is

(a) before using the [Train X button](#Train_X): the agent in the X-agent select box (which is the first element of Types::GUI\_AGENT\_INITIAL, currently MCTS),   
(b) after using the [Train X button](#Train_X): the last trained X-Player.

Different after states **s** can be set by the user in the GameBoard board buttons

* for **TicTacToe**: clicking a button moves it through the circle [ ] -> X -> O -> [ ] -> …).
* for **Hex**: clicking a tile sets a black (X) or white (O) piece there
* for **2048**: clicking one of the four action buttons (up, right, …) performs this action, adds a new random tile and shows the values V(a’) of the then possible actions a’.

The value of V(**s’**) for each allowed successor **s’** of **s** is displayed on the GameBoard in a game-specific way.

### Params X, Params O

Display the multi-tabbed Params window and set parameter in any of the tabs ([TD params](#TD_params) and so on, see below). The parameters are fetched from this multi-tabbed window when one of the train buttons, Train X, Train O, or MultiTrain is pressed. This is for the trainable agents. For the non-trainable agents, the parameters are fetched from this multi-tabbed window when Quick Eval, Compete, Multi Compete, or Save Agent (Arena menu) are issued or when buttons [Play](#Play) or [InspectV](#InspectV) are pressed.

### Train X

Fetch the parameter settings from the multi-tabbed Params window. Set the X agent according to the X combo box and train it for “Train games” games.

During training: an Evaluator with mode [Quick Eval Mode](#QuickEvalMode) is called every [NumEval](#NumEval) training games and its result is shown in a JFreeChart XY-plot.

During training: If [StopTest](#StopTest)>0 and [StopEval](#StopEval)>0, the same Evaluator is called every StopTest training games. If the Evaluator signals “Training goal reached” (i.e. sufficient good play for a sufficient long period, see [Other params](#Other_params) for more details), the training is stopped prematurely.

After training, the trained player is evaluated (Evaluator.eval()). This is for example in the case TicTacToe:

“Success against random” = average success rate when playing 100 games against RandomPlayer, both as X and O (optimum: 1.0),  
“Success agains minimax” = success rate when playing an X- and an O-game against MinimaxPlayer (optimum: 0.0, i.e. always tie).

Note that the success rate becomes negative, when the other player predominantly wins.

### Train O

Same as [Train X](#Train_X), but for the O-player.

### MultiTrain

Same as [Train X](#Train_X), but perform “Agents trained” training runs and report the average success.

## Competition menu

This menu is only relevant for 2-player games.

### Single Compete

Make a competition “X vs O” consisting of “Games/Comp” games and report results.

### Swap Compete

Swap the roles of X and O, i. e. make a competition “O vs X” consisting of “Games/Comp” games and report results.

### Multi-Competition

Perform “Competitions” competitions “X vs O” and report results. The agents (if trainable) are trained anew before each competition.

## TD params

Parameter for Temporal Difference Learning (for [TDS](#TDS) player, [TD-Ntuple](#TDS_NTuple) agent):

* **Alpha init**: initial learning rate
* **Alpha final**: final learning rate
* **Epsilon init**: initial random move rate (clipped to allowed range [0, 1])
* **Epsilon final**: final random move rate (clipped to allowed range [0, 1])
* **Lambda**: eligibility trace parameter
* **Gam****ma**: discount factor in range [0, 1]
* **Network type**:[linear] the output activation is either a linear function of the (generalized) input features or a backpropagation network with one hidden layer of size 15.
* **Output sigmoid**:[without]should the output unit be with a sigmoid? If“with”, then the Fermi function  
    
  is used as sigmoid in case of agent type **TDS**.  
  If the agent is of type [TD-Ntuple](#TDS_NTuple), then there is always an output sigmoid active, and this is the Tangens Hyperbolicus:
* **Normalize**: Ifchecked, then each game score returned from the state observer is normalized to a range appropriate for the output sigmoid, that is range [0,1] in the case of Fermi function (TDS) and [-1,1] in the case of Tangens Hyperbolicus ([TD-Ntuple](#TDS_NTuple)).

If the value function is approximated by a neural network, the effective learning rate for the input-to-hidden weights is Alpha divided by the input-fan-in (size of input layer) and the learning rate for the hidden-to-output weights is Alpha divided by the hidden-fan-in (size of the hidden layer).

### Feature sets

Feature sets for [TDS](#TDS) player in the case of game **TicTacToe**:

* **0**: singlets/doublets/triplets for “self” and “opponent”
* **1**: singlets/doublets/triplets for “X” and “O”
* **2**: singlets/doublets + diversity + crosspoints for “X” and “O”
* **3**: same as **2** + the 9 “raw” board positions
* **4**: same as **2** + occupation midpoint, occupation corner
* **5**: same as **2** + …
* **9**: the 9 “raw” board positions

## MCTS & MCTSE params

Parameter for [MCTS](#MCTS) agent (class MCTSPlayer):

* **Iterations**: [1000] how many rollouts are performed
* **K[UCT]**: [sqrt(2)] balances exploitation and exploration in the UCT formula
* **Tree Depth**: [10] the maximum MCTS tree depth
* **Rollout Depth**: [10] the maximum rollout depth (how many plys)

MCTSE has the same parameters plus **Max Nodes**, the maximum number of allowed nodes (Expectimax nodes) in the tree

## Other params

During or after training an agent, this agent can be evaluated by an evaluator (see below). If such an evaluator signals “success” (for a long enough training period of StopEval training games), then training might be stopped prematurely.

Settings:

* **Qui****ck Eval Mode**: An Evaluator with this mode is used in ‘Quick Evaluation’, in training and in multi-training. This is the evaluator with the [StopEval](#StopEval) test to end training prematurely. This is as well the evaluator whose performance during training is shown in the JFreeChart XY-plot every [NumEval](#NumEval) training games.
* **Train Eval Mode**: If different from [Quick Eval Mode](#QuickEvalMode), another evaluator is constructed after training and multi-training. It is evaluated in parallel to assess the strength of an agent after training from a different perspective. If Train Eval Mode is identical to [Quick Eval Mode](#QuickEvalMode), no second evaluator is constructed.
* **N****umEval**: [100] after every NumEval training games the performance of the trained agent is evaluated (success against [Minimax](#Minimax), success against [Random](#Random)) and the success against [Minimax](#Minimax) is plotted in a JFreeChart window. Choose higher values for NumEval to speed up training.
* **Sto****pTest**: [0] after every StopTest training games an Evaluator is called to see if we can stop training prematurely. If 0, this Evaluator is never called and so training is never stopped prematurely.
* **Sto****pEval**: [0] number of *consecutive* games an Evaluator has to signal “success” before training is stopped prematurely. If 0, training is never stopped prematurely.
* **Minimax Depth**: [10] the maximum tree depth (recursion depth) of the Minimax agent.
* **Minimax Hash**: [true] use hash map to store already visited states

## Available evaluators

### TicTacToe

* mode 0: competition (100 games) against RandomAgent
* mode 1: competition (1 game) against Minimax
* mode 2: competition (10 games) against Minimax from 10 different start states
* mode 9: measure rate of correct decisions and game function delta on a set of 24 states

An evaluation of EvaluatorTTT is termed a “success”, if its return value is above the threshold m\_thresh (currently m\_thresh = {0.8,-0.15,-0.15, 0.85} in source code).

### Hex

* mode 0: competition (5 games) against MCTS

### 2048

* mode 0: the score in 50 (ConfigEvaluator.NUMBEREVALUATIONS) games

## Help

### Show Help File

Toggle the display of this help text in a HTML window.

### Help File in Browser

The same in a browser window.

### Show TR-GBG.pdf

Show the document “**The GBG Class Interface Tutorial: General Board Game Playing and Learning**” by Wolfgang Konen.

1. The former agent “**CMA****-ES**: TD Learning agent using neuroevolution with CMA-ES as approximator for the value function” is currently not available. [↑](#footnote-ref-1)