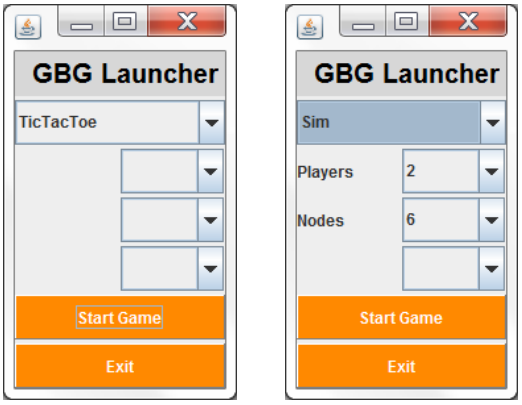
Help File for GBG GUI

CONTENTS

GBG Launcher

When starting **GBGLaunch.jar**, a small launcher window appears which allows to select any of the playable games:



Depending on the game selected in the upper checkbox, which is one out of

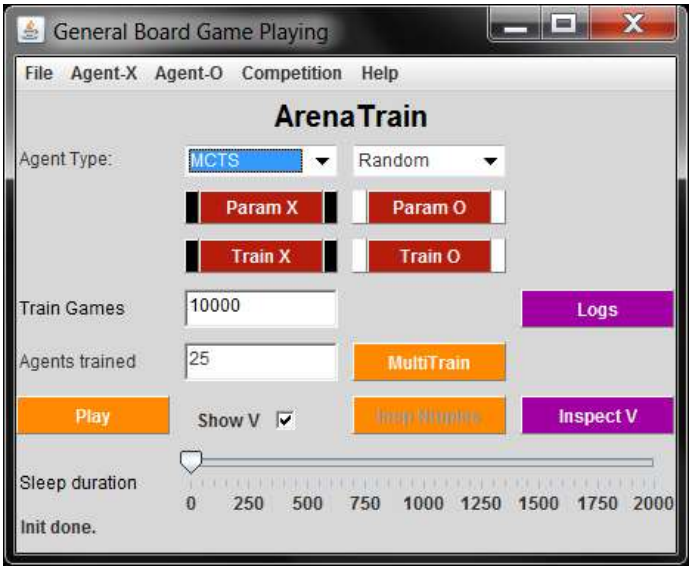
- Tic-Tac-Toe (Noughts and Crosses)
- Nim (arbitrary number and size of heaps)
- Hex (arbitrary board sizes)
- 2048 (non-deterministic)
- Connect Four (Connect-4, Four-in-a-Row, Captain's Mistress)
- Othello (Reversi)
- Sim (Ramsey game, a.k.a. Hexi, arbitrary # nodes, 2- and 3-player variants)
- Rubik's Cube (beta),

the three lower checkboxes will or will not allow to select **scalable parameters**. For example, in the case of Sim, there will be two scalable parameters: the number of players (2 or 3) and the number of nodes (arbitrary in principle, constrained in practice by the options given in the checkbox).

Press **Start Game** to start the selected game: Arena Train and the game-specific game board will appear and the GBG Launcher becomes invisible. When exiting the selected game, the GBG Launcher will appear again.

See GBGBatch for the batch launcher.

Arena & Arena Train Elements



Agent Type

The select boxes allow to select one of the following agent types for every player:

- **TDS**: TD Learning agent according to [Sutton&Bonde 1993] with a linear net or backprop net (one hidden layer) as approximator for the value function, needs user-defined features (class TDAgent).
- **TD-Ntuple-2, TD-Ntuple-3**: TD-Learning agent using n-tuples: a form of reinforcement learning, new variant with afterstate and new lambda-mechanism [Jaskowski16] (class TDNTuple3Agt).
- **Sarsa**: SARSA-learning (state-action-reward-state-action), a form of reinforcement learning, with n-tuples.
- **MC**: Monte Carlo
- **MCTS**: Monte Carlo Tree Search
- **MCTS Expectimax**: Monte Carlo Tree Search for nondeterministic games
- **Max-N**: Tree search agent, generalization of the minimax principle to N players for deterministic search (w/o Alpha-Beta-search). Realizes perfect play for TicTacToe, but may ‘explode’ for more complex games.
- **Expectimax-N**: the generalization of Max-N to N players for nondeterministic games.
- **Random**: random playing agent
- **Human**: human player

Depending on the game chosen there may be additional agents in the select box which are specific to the game in question

- **AlphaBetaAgent** for Connect-4,
- **BoutonAgent** for Nim,
- **Edax2, HeurPlayer, BenchPlayer** agents for Othello,
- **Davi-X** agents for RubiksCube.

Most of these agents provide (near-) perfect-playing opponents for evaluations. In case of Othello there are also the non-perfect agents **HeurPlayer** and **BenchPlayer**.

Each of the above agents (except Human) can be augmented with a wrapper agent to perform n-ply look-ahead.

Play

Play a game with the agents currently selected as "X" or "O" in the Agent Type 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, the user has to make the appropriate moves on the game board.

The starting player is always "X" (2-player games) and "Agent-0" for N-player games (N>2). The game starts from the initial start board (in many games the empty board). To start from other boards see InspectV.

If checkbox **Show V** is selected, the game board will display the game values (or colors) for the agent who just **made** the last move. Thus, the values are *not* a hint for the next move, but a sort of explanation of the last move and how confident the agent was in his decision. (Note the difference to InspectV, where the game values shown are those of the available moves for the **current** state, i.e. for the player who **has to** move.)

InspectV

Inspect the value function V(**s'**) of the X-Player (Agent-0 for N-player games with N>2). The X-Player is

(a) before using the Train X button: 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: the last trained X-Player.

Different states **s** can be set by the user in the action buttons in GameBoard:

- for **TicTacToe**: clicking a tile sets a black (X) or white (O) piece there,
- 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 values or colors shown on the GameBoard (in a game-specific way) are shown irrespective of the state of checkbox **Show V**. They show V(**s'**) for each allowed successor states **s'** of the current state **s**. Thus, the values provide really a hint which is the best next move to play (in the view of the X-agent)

When in INSPECTV mode, all other buttons (except InspectV and Play) are disabled.

- Another click on InspectV ends the INSPECTV mode and enables again all other buttons.
- A click on Play starts playing a game from the actual board, as configured by InspectV. This allows to test how an agent performs when it starts from a position different from the default initial position (empty board).

Params X, Params O

Display the multi-tabbed Params window and set parameter in any of the tabs (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, or Save Agent (Arena menu) are issued or when buttons Play or InspectV are pressed.

Train X

Fetch the parameter settings from the multi-tabbed Params window. Construct the X agent according to the X combo box and train it for "Train games" episodes.

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

During training: If StopTest>0 and 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 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 when playing 100 games against RandomPlayer, both as X and O (optimum: 1.0),
> "Success agains minimax" = success when playing an episode in both roles X and O against MinimaxPlayer (optimum: 0.0, i.e. always tie).

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

Train O

Same as Train X, but for the O-player.

MultiTrain

Same as Train X, but perform "Agents trained" training runs w/o plotting to the line chart (this is useful for running a training completely in background and to average performance over random fluctuations). Store all information needed to construct such a line chart later in agents/<gameDir>/csv/multiTrain.csv. (There are several R-script templates in directory resources/R_plotTools for visualizing the results from one or several files multiTrain.csv.) Report the average success in console. Return the last trained agent as the agent for X.

Agent menu

For each agent there is an Agent-* menu with the following entries:

Load Agent

Save Agent

Agents are stored in and loaded from

agents/<gameDir>/<agentName>.agt.zip

<gameDir> is either the name of the game or it is the name of the game plus a subdirectory characterizing the game subtype, e.g. Hex/04 in the case of a 4x4 Hex game.
<agentName> is an arbitrary name characterizing the agent.

An agent is saved and loaded with **all** parameters that characterize it. On load, these parameters are automatically restored in the Param Tabs.

Quick Evaluation

Run an evaluation with the evaluator in mode QuickEvalMode

Competition menu

This menu is not relevant for 1-player games.

Single Compete

Make a competition "X vs O" consisting of "Games/Comp" game episodes and report results.

Swap Compete

(only 2-player games) Swap the roles of X and O, i. e. make a competition "O vs X" consisting of "Games/Comp" game episodes and report results.

Compete In All Roles

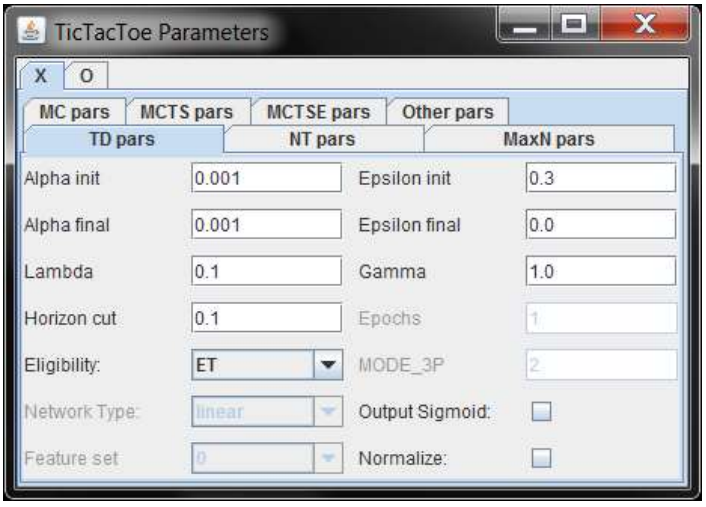
Make a competition where each player takes in turn all roles. Example for N=3 players:

- P0 as 1st, P1 as 2nd, P2 as 3rd
- P1 as 1st, P2 as 2nd, P0 as 3rd
- P2 as 1st, P0 as 2nd, P1 as 3rd

For each role, play "Games/Comp" episodes and report results.

Param Tabs

The parameters for all agents are set in the hierarchical multi-tabbed Params window:



Each player (here: "X" and "O") has a multi-tabbed sub-window for setting the parameters for specific agents

TD Params

Parameter for Temporal Difference Learning (for TDS, TD-Ntuple-X or Sarsa agent):

- **Alpha init**: initial learn step size α_i in first training episode.

- **Alpha final**: final learn step size α_f in last training episode. The change from Alpha init to Alpha final is a geometric (exponential) slope: $\alpha(t) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^t$, where t is a real-valued training time parameter: 0 in the first episode and 1 in the last episode.
- **Epsilon init**: initial random move rate in first training episode.
- **Epsilon final**: final random move rate in last training episode. The change from Epsilon init to Epsilon final is a linear slope. A random move is done, if a uniform random number $\in [0,1]$ is smaller than Epsilon. Epsilon may have values outside $[0,1]$. If for example $\varepsilon_i = 0.3,\ \varepsilon_f = -0.3$ then the random move rate will drop below 0 after half of the total training episodes. This means that no random moves will happen after this point in time.
- **Lambda**: eligibility trace parameter λ. Usually, $\lambda \leq \gamma$ should hold.
- **Horizon cut**: (only enabled if $\lambda > 0$) the horizon for eligibility traces.
- **Eligibility**: (only enabled if $\lambda > 0$) the eligibility mode (ET: normal, RESET: reset trace on random move)
- **Gamma**: discount factor γ in range [0, 1]
- **Network type** (for TDS only): [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. For TD-Ntuple-X or Sarsa, the network type is always linear.
- **Output sigmoid**: [not checked] should the output unit be with a sigmoid? If checked, then the Fermi function

  $$\sigma(y) = \frac{1}{1+e^{-y}}$$

  is used as sigmoid in case of agent type TDS.
  If the agent is of type TD-Ntuple-X or Sarsa, then – if checked -- the output sigmoid is Tangens hyperbolicus:

  $$\sigma(y) = \tanh(y)$$
- **Normalize**: If checked, 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-X or Sarsa).
- **MODE_3P** (for TD-Ntuple-2 only): Different update schemes to extend the normal TD update process to arbitrary N-player games
  - **0**: Update N independent value function $V(s_t|p^{(i)})$ for all i=0,…,N-1 in each time step t. Works for arbitrary N, but weaker results for 2-player games than MODE_3P=1 or =2 (unwanted crosstalk).
  - **1**: Update only for $V(s_t|p_t)$ and make an **N-ply** evaluation.
  - **2** [default]: Mixed version to take the best out of MODE_3P=0 and =1:
    - **1-player games**: take MODE_3P=0
    - **2-player games**: take MODE_3P=1 with **1-ply** evaluation and use the symmetry $V(s_{t+1}|p_t) = -V(s_{t+1}|p_{t+1})$
    - **N-player games with N>2**: take mode=0 (and live with the cross-talk)
- **Epochs** (for TDS only): Accumulate gradient for 'Epochs' iterations, then update weights.

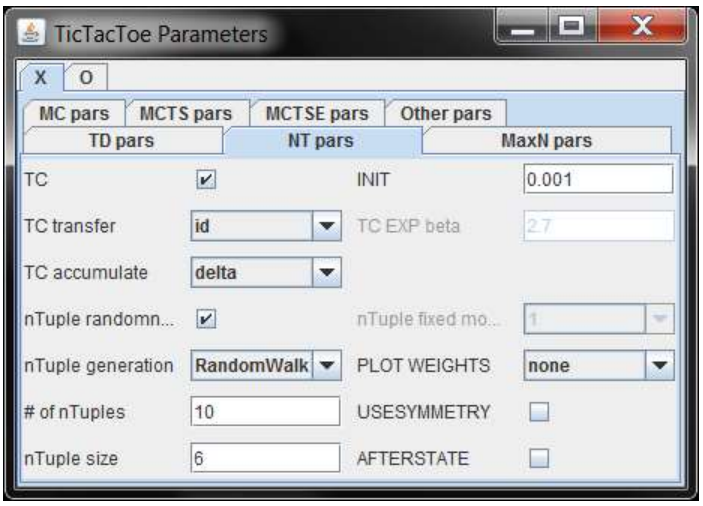
If the value function is approximated by a neural network, the effective learning rate for the input-to-hidden weights is Alpha divided by the input-fan-in (size of input layer) and the learning rate for the hidden-to-output weights is Alpha divided by the hidden-fan-in (size of the hidden layer).

TD Params: Feature Sets

Feature sets for 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

NT Params



Parameter for n-tuple specification and for Temporal Coherence (used by agents TD-Ntuple-X and Sarsa):

- **TC**: Temporal Coherence on/off
- **INIT**: (only if TC) TC train counters are initialized with random numbers from [0,INIT]
- **TC transfer**: (only if TC) transfer function [id] identity or [TC EXP] exponential
- **TC EXP beta**: (only if TC EXP) exponential parameter
- **TC accumulate**: (only if TC) what to accumulate in TC counters: [delta] or [rec wght change] recommended weight change.
- **AFTERSTATE** (only relevant for *nondeterministic* games, e.g. 2048): If checked, use the afterstate logic as described in [Jaskowski16]: The argument s’ in V(s’) is the state after an action, **prior** to adding the non-deterministic element. If not checked, the argument s for V(s) is the next state (including the non-deterministic element). For *deterministic* games, AFTERSTATE is disabled, since afterstate and next state are the same.
- **nTuple randomness**: if checked, construct random n-tuples. If not checked, use fixed n-tuples.
- **nTuple generation**: RandomWalk (connected n-tuples), RandomPoints (arbitrary points on the board)

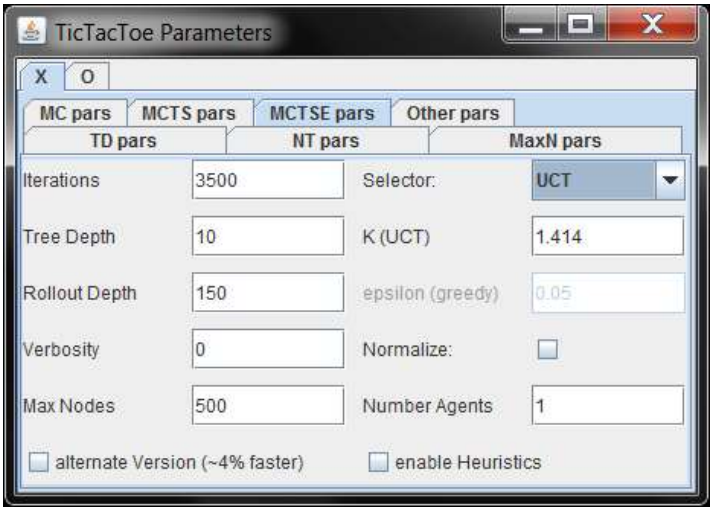
- **# of nTuples**: how many random n-tuples to generate
- **nTuple size**: every generated n-tuple has a size 2,…,nTupleSize
- **nTuple fixed mode**: select one of the possible modes, each is connected with a fixed set of n-tuples (see tooltips)
- **USESYMMETRY**: If checked, then during training and retrieval each board state activates also all equivalent board states (those connected by the symmetries of the game, e.g. four rotations*two reflections in the case of 2048). If not checked, do not use the equivalent board states
- **nSym** (only relevant if USESYMMETRY is checked): do not use all symmetries (may be too many for some games), but only nSym randomly selected symmetries. The default nSym=0 means “use all symmetries”.

MC Params

Parameter for MC agent (class MCAgent) [defaults in brackets]:

- **Iterations**: [1000] how many rollouts are performed
- **Rollout Depth**: [20] the maximum rollout depth (how many plys)
- **Number agents**: [1] use an ensemble of this number of MC agents and base the action decision on majority vote

MCTS & MCTSE Params



Parameter for MCTS agent (class MCTSAgentT) [defaults in brackets]:

- **Iterations**: [1000] how many rollouts are performed
- **Selector**: [UCT] which tree policy to follow, which nodes to select: UCT = Upper Confindence Bound for Trees, eps-greedy = epsilon greedy, roulette wheel = select nodes with probability in proportion to their value
- **K[UCT]**: [1.414] (only if Selector = UCT) balances exploitation and exploration in the UCT formula
- **epsilon greedy**: (only if Selector = UCT) select with probability epsilon a random node, else select the node with the best value
- **Tree Depth**: [10] the maximum MCTS tree depth
- **Rollout Depth**: [10] the maximum rollout depth (how many plys)
- **Verbosity**: [0] 0=silent, 1=one line on System.out, 2, 3, … = one line for each child, grand-child, …

- **Normalize**: if checked, normalized the rollout value to range [0,1]. This requires to know what the maximum rollout value is. In case of 2048, the theoretical rollout maximum would be too high in most cases, so a maximum score is estimated online, specific to the state in question.

MCTSE (for MCTS Expectimax agent) has the same parameters **plus**:

- **Max Nodes**, the maximum number of allowed nodes (Expectimax nodes) in the tree
- **Number Agents**: the number of agents used for majority vote
- **alternate version**: (recommended not to check)
- **enable heuristic**: (specific for 2048 only, recommended not to check)

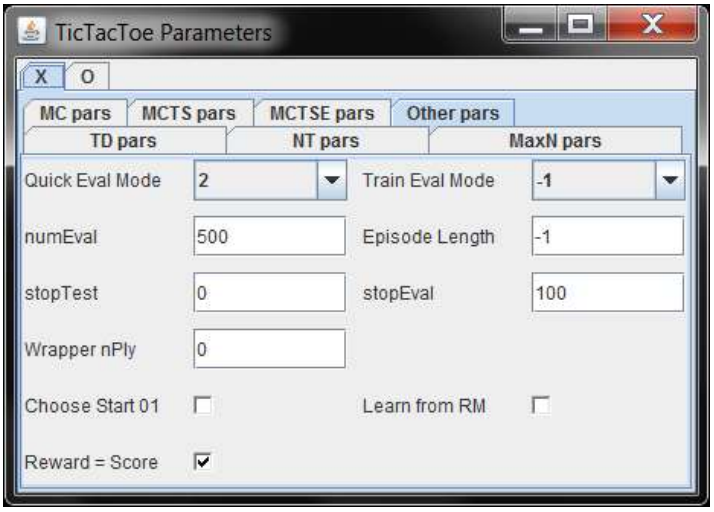
MaxN Params

Parameter for Max-N, Expectimax-N or wrapper agents :

- **Tree Depth**: [10] the maximum tree depth (recursion depth) of the agent.
- **Max-N Hashmap**: [true] use hash map to store already visited states (only Max-N). ExpectimaxN, MaxNWrapper and ExpectimaxNWrapper have NO hash map.

Other Params

Upper part: Parameters relevant for all agents (evaluation + wrapping).
Lower part: Parameters relevant for all trainable agents (training).



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:

- **Quick Eval Mode**: An evaluator with this mode is used in Quick Evaluation, in training and in multi-training. See **tooltip text** of the select box for a short description of available modes. This is the evaluator with the 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 training games.
- **Train Eval Mode**: If different from Quick Eval Mode, another evaluator is constructed in 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, no second evaluator is constructed.

- **NumEval**: [100] after every NumEval training games the performance of the trained agent is evaluated (success against Minimax, success against Random) and the success against Minimax is plotted in a JFreeChart window. Choose higher values for NumEval to speed up training.
- **StopTest**: [0] after every StopTest training games, the Quick Eval Mode 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.
- **StopEval**: [0] number of *consecutive* games that the Quick Eval Mode evaluator has to signal "success" before training is stopped prematurely. If 0, training is never stopped prematurely.
- **Episode Length**: [-1] During training, stop a game prematurely, if the episode length is reached. If -1, the game is never stopped prematurely.
- **Wrapper nPly**: [0] During Play, Quick Evaluation, Competition or InspectV: use either Max-N or Expectimax-N with this nPly as a wrapper around the actual agent. Which of both wrappers is used depends on the game: Max-N for deterministic, Expectimax-N for nondeterministic games. The wrapper constructs a tree of depth nPly and once the tree leaves are reached, it asks the wrapped agent for a game value estimate. This increases the performance in most cases, but be cautious with nPly>3: Depending on the branching factor of the game, this may lead to exploding computation times. If set to 0, no wrapper is used.

Parameters relevant for training (all trainable agents):

- **Choose Start 01**: If not checked, each training episode is started from the default (empty) game board. If checked, it is in 50% the empty game board and the other 50% are distributed equally on one of the possible 1-ply successors of the empty board. Increases the exploration.
- **Learn from RM**: If not checked, do not learn during training when a random move occurs. If checked, learn from each move (random action or not).
- **Reward = Score**: The boolean rewardIsGameScore is passed to StateObservation's getReward(rewardIsGameScore) and. Depending on the boolean value, getReward returns either the usual game score or another (game-specific coded) reward.

Logs

During game play, played episodes can be logged. Logged episodes can be restored and re-played. Episodes are stored in

logs/<gameDir>/<game>_<date>_<time>.gamelog

See here for explanation of <gameDir>.

Button **Logs** opens the Log Manager Window:

Options Menu

- **Logging enabled**: [true] if checked, log every episode started via Play
- **Advanced logging**: [true] if checked, enable advanced logging, i.e. the possibility to restore from a temporary gamelog when the main program has crashed before the logging session was closed normally, see LogManager-QQ2.pdf (in German) for details.
- **Verbose**: [true]
- Compile temporary gamelog
- Delete temporary gamelogs

Load Menu

Load Gamelog opens a file chooser for logs/<gameDir>.

After selecting a `gamelog` file, the episode stored in that file is loaded and can be replayed via buttons **Advance** and **Revert**, or you enter a move number and select it with **Jump to Move**. In either case, the gameboard window will display the selected state of that episode.

Available evaluators

Each game has an Evaluator class which can construct evaluators in different modes. (See also tooltip texts when hovering over the evaluator select boxes.)

TicTacToe

- mode -1: disable evaluation.
- mode 0: competition (100 games) against RandomAgent
- mode 1: competition (1 game) against Max-N
- mode 2: competition (10 games) against Max-N from 10 different start states
- mode 11: competition against TDReferee.agt.zip (if this file is available in agents/<gameDir>), 10 different start states.

An evaluation 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} for the different modes in source code).

Hex

- mode -1: disable evaluation.
- mode 0: against MCTS
- mode 1: against Random
- mode 2: against Max-N (depth 10)
- mode 10: against MCTS, different starts.
- mode 11: against TDReferee.agt.zip (if available in agents/<gameDir>), different starts.

Some details:

- MCTS configuration (mode 0 and mode 10): 10^{K-1} iterations with K= min(HexConfig.BOARD_SIZE,5), treeDepth=10, and rolloutDepth=200.
- mode 0 details: Plays 10 episodes. MCTS plays 2nd. Strong evaluator for board sizes up to and including 5x5. No guarantees for 6x6 or higher. Tends to require a lot of memory for 7x7 and up.
- mode 1 details: Plays 100 episodes. Random plays 2nd. Very weak but fast evaluator.

- mode 2 details: Plays 100 episodes. Max-N plays 2nd. Strong evaluator if applicable. But **only applicable for up to 4x4** board sizes (otherwise memory & time requirements too large), and even for 4x4 boards it takes quite a while on first pass.
- mode 10 and mode 11 details: Plays 10 episodes for each different start state coded in HexConfig.EVAL_START_ACTION[N][] where N is the size of the Hex board. All start states are winning boards. The agent to be evaluated makes the first move. More statistically reliable than the mode-0 evaluator, but the remarks there about MCTS apply as well. Evaluation time in mode-10 is S times longer than in mode-0 with S = length(HexConfig.EVAL_START_ACTION[N][]).

2048

- mode -1: disable evaluation.
- mode 0: use Evaluator2048: average score from 50 (ConfigEvaluator.NUMBEREVALUATIONS) evaluation episodes
- mode 1: use Evaluator2048_BoardPositions: load a large set of board positions from file gameStates.ser, analyze them when applying MC and MCTSE agents to them.
- mode 2: use Evaluator2048_EA: perform a CMA-evaluation of the heuristics (see [Kutsch2017])

Connect-4

- mode -1: disable evaluation.
- mode 0: against MCTS, best is 1.0
- mode 1: against Random, best is 1.0
- mode 2: against Max-N, best is 1.0
- mode 3: against AlphaBeta, best is 1.0
- mode 4: against AlphaBeta, different starts, best is 1.0
- mode 5: against AlphaBetaDL (AlphaBeta with **distant losses**), best is 1.0
- mode 10: against MCTS, different starts, best is 1.0
- mode 11: against TDReferee.agt.zip (if available in agents/<gameDir>), different starts

All evaluators let the agent to evaluate start in that player role where a win is possible. Therefore, the best evaluation result is always 1.0.
AlphaBeta is an agent special for Connect-4 which plays perfect. But it will make random moves if it is in a losing position, i.e. it gives up early (and does not test whether the evaluated agent plays perfect *throughout*).
AlphaBetaDL is an agent even harder to beat: If in a losing position, it selects among all possible moves that move that postpones the loss as far in the future (in the distance) as possible.

Nim

- mode -1: disable evaluation.
- mode 0: against Random, best is 0.9 – 1.0 (depending on heap config)
- mode 1: against Max-N (depth 15), best is 1.0
- mode 2: against Max-N (depth 15), different starts, best is 1.0
- mode 3: against MCTS (iterations 1000), best is 1.0
- mode 4: against MCTS (iterations 1000), different starts, best is 1.0

- mode 11: against TDReferee.agt.zip (if available in agents/<gameDir>), different starts

Both agents play both roles for every start position.
If the agent to evaluate plays perfect, but the evaluator is weak, the best result is 1.0.
If the evaluator plays perfect, the evaluation result is in range [-1.0, 0.0].

Othello

- mode -1: disable evaluation.
- mode 0: against Random, best is 1.0
- mode 1: against Max-N (depth 4), best is 1.0
- mode 2: against MCTS (iterations 1000), best is 1.0
- mode 9: against BenchPlayer, best is 1.0
- mode 10: against HeurPlayer, best is 1.0
- mode 11: against TDReferee.agt.zip (if available in agents/<gameDir>), expected 0.0
- mode 19: against BenchPlayer, different starts, best is 1.0
- mode 20: against HeurPlayer, different starts, best is 1.0
- mode 21: against TDReferee.agt.zip (if available in agents/<gameDir>), different starts, expected 0.0

Here, *different starts* means the following: There is a list of 244 start states which are 4 ply away from the default start state. In evaluator mode 19, 20 and 21, each evaluator plays for each of these start states one episode in both roles against the evaluated agent. If TDReferee.agt.zip is a (near-)perfect playing agent, the best result in mode 11 or 21 is expected to be 0.0.

Sim

- mode -1: disable evaluation.
- mode 0: against Random, best is 0.9 – 1.0 (depending on number of nodes)
- mode 1: against Max-N, best is 0.0
- mode 2: against MCTS, best is 0.0

All evaluators (except mode 0) have as best result 0.0, because each agent plays in all roles. (This assumes that the evaluator plays perfect, which is not necessarily the case.) If the evaluator plays perfect, but the evaluated agent not, the evaluation result is negative in range [-1.0, 0.0).
Sim has a 2- and a 3-player variant. In the 3-player variant, two independent agents of the mode-type are created and they play in all roles against the evaluated agent.

Tournament System menu

This allows to start and inspect tournaments between a group of agents.

Start Tournament System

Start a tournament and analyze results.

Load & Show Results from Disk

Load saved tournament results from disk and analyze them.

GBGBatch

The batch facility **GBGBatch.jar** offers different possibilities to start (time consuming) training runs as batch runs. An agent with all its parameter settings is loaded from disk and one or multiple training runs for this agent are started.

Syntax:

```
GBGBatch gameName n agentFile [ trainNum maxGameNum csvFile scaPar0 scaPar1 scaPar2 ]
```

with

- gameName: name of the game, suitable as subdirectory name in the `agents` directory
- n = 1, 2 or 3; to call either multiTrain, multiTrain with AlphaSweep, or with LambdaSweep, resp. Default: 1.
- agentFile: filename of agent, e.g. "tdntuple3.agt.zip". This agent is loaded from `agents/<gameDir>`, where `<gameDir>` is either the name of the game or the name of the game plus a subdirectory characterizing the game subtype, e.g. Hex/04 in the case of a 4x4 Hex game.
  The loaded agent specifies the agent type and all its parameters for training.
- (optional) trainNum: how many agents to train (default -1)
- (optional) maxGameNum: maximum number of training episodes (default -1)
- (optional) csvFile: filename for CSV results (defaults: "multiTrain.csv", "multiTrainAlphaSweep.csv", "multiTrainLambdaSweep.csv", depending on n)
- (optional) scaPar0,1,2: scalable parameters

If trainNum or maxGameNum are -1, their respective values stored in agentFile are taken.

scaPar0,1,2 contain the scalable parameters of a game (if a game supports such parameters). Example: The game Hex has the board size (4,5,6,...) as scalable parameter scaPar0. If no scalable parameters are given as command arguments, the defaults from `GBGBatch.setDefaultScaPars(String)` apply.

Side effect: the last trained agent is stored to <csvName>.agt.zip, where <csvname> is `args[5]` w/o ".csv".

AlphaSweep: For each $\alpha \in$ {`1.0, 2.5, 3.7, 5.0, 7.5, 10.0`} we conduct trainNum runs with $\alpha_{init} = \alpha_{final} = \alpha$.

LambdaSweep: For each $\lambda \in$ {`0.00, 0.04, 0.09, 0.16, 0.25`} we conduct trainNum runs.

Help

Hints for installation and configuration of GBG are found in the GBG Wiki: https://github.com/WolfgangKonen/GBG/wiki/Install-and-Configure.

Show Help File

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

Help File in Browser

Display this help text in a browser window.

Help File as PDF

Display this help text in a PDF viewer.

Game Rules

Display the rules of all GBG games in a browser window.

Game Rules as PDF

Display the rules of all GBG games in a PDF viewer.

Show TR-GBG.pdf

Show the document "**The GBG Class Interface Tutorial V2.1: General Board Game Playing and Learning**" by Wolfgang Konen.

NOTE: When starting GBG from JAR (GBGLaunch.jar), the display of HTML sources will only show the base HTML file, not any subsequent resources (images, formulas). To show all elements, use either PDF display, which is always in *one* file, or start GBG from Java sources.