General Board Game Playing as Educational Tool for AI Competition and Learning

Wolfgang Konen
Computer Science Institute
TH Köln – University of Applied Sciences
Germany
Email: wolfgang.konen@th-koeln.de

***Abstract*—We present GBG, a new general board game playing and learning framework. GBG defines the common interfaces for board games, game states and their AI agents. It allows to run competitions of different agents on different games. It standardizes those parts of board game playing and learning that otherwise would be tedious and repetitive parts in coding. GBG is suitable for arbitrary 1-, 2-, ..., N-player board games. GBG aims at the educational perspective, where it helps students to start faster in the area of game learning. GBG aims as well at the research perspective by collecting a growing set of games and AI agents to assess their strengths and generalization capabilities in meaningful competitions. First successful educational and research applications are reported.** [1]

I. Introduction

A. Motivation

General board game (GBG) playing and learning is a fascinating area in the intersection of machine learning, artificial intelligence (AI) and game playing. It is about how computers can learn to play games not by being programmed but by gathering experience and learning by themselves (self-play). Recently, AlphaGo Zero [1] and AlphaZero [2] have shown remarkable successes for the games of Go and Chess with high-end deep learning and stirred a broad interest for this subject.

A common problem in game playing is the fact, that each time a new game is tackled, the AI developer has to undergo the frustrating and tedious procedure to write adaptations of this game for all agent algorithms. Often he/she has to reprogram many aspects of the agent logic, only because the game logic is slightly different to previous games. Or a new algorithm or AI is invented and in order to use this AI in different games, the developer has to program instantiations of this AI for each game.

GBG is a recently developed software framework consisting of classes and interfaces which standardizes common processes in game playing and learning. If someone programs a new game, he/she has just to follow certain interfaces described in the GBG framework, and then can easily use and test *all* AIs in the GBG library on that game.

The first motivation for GBG was to facilitate the entry for our Bachelor and Master students into the area of game learning (*educational perspective*): Students at our faculty are often very interested in game play and game learning. Yet the time span of a project or a Bachelor (Master) thesis is usually too short to tackle both game development and AI development 'from scratch'. The GBG framework allows a quicker start for the students, it offers over time a much larger variety of AIs and games and it allows students to focus on their specific game and AI research.

The second motivation is to facilitate for researchers the competition between AI agents on a variety of games to drive the research for versatile (general) AI agents. This has some resemblance to General Game Playing (GGP), but there are also important differences (see Sec. II-A).

The third motivation is, once we have a decent number of games together, to set up real competitions for researchers which hand in their AIs to compete on a number of games. At present, to the best of our knowledge, there are no such competitions (besides GGP, see Sec. II-A) with sophisticated AIs tackling 1-, 2-, ... N-player games *at the same time*. Such competitions will be done when a greater variety of board games is included in GBG.

Last but not least, a competition with a large variety of agents on a game helps to solve the often non-trivial question on how to evaluate the real strength of game-playing agents in a fair manner.

The rest of this document is structured as follows: After a short (and probably incomprehensive) summary of related work in Sec. I-B, Sec. II gives an overview of GBG: classes, agents and games currently implemented in GBG. Sec. III discusses some early results obtained with this framework. Sec. IV concludes.

B. Related Work

Epstein [3] presented with *Hoyle* an early general board game playing program. It learned to play 18 diverse board games, where its strategic principles are general and not game specific. The software implementation of *Hoyle* is not further described in [3].

There is the discipline **General Game Playing (GGP)** [4], [5] which has a long tradition in artificial intelligence: A GGP competition organized by the Stanford Logic Group is held annually at the AAAI conferences since 2005 [6]. Given the game rules written in the so-called *Game Description*

[1]This work has been submitted to *IEEE Transactions on Games* for possible publication.

Name	Type	Description
StateObservation	IF	defines an abstract board game: state, rewards, available actions, game rules, ...
StateObsNondeterministic	IF	derived from *StateObservation*: additional methods for nondeterministic games
GameBoard	IF	game board GUI: display states & action values, HCI for human players, ...
PlayAgent	IF	the agent interface: select next action, predict game value, perform training, ...
Arena	AC	meeting place for agents on an abstract game: load agents, play game, evaluation, competition
ArenaTrain	AC	derived from *Arena* for additional capabilities: parametrize, train, inspect & save agents

Table I
THE MAIN CLASSES OF GBG. 'TYPE' IS EITHER INTERFACE (IF) OR ABSTRACT CLASS (AC).

Language (GDL, [7]), several AIs enter one or several competitions. As an example for GGP-related research, Mandziuk et al. [5] propose a universal method for constructing a heuristic evaluation function for any game playable in GGP. A GGP learning framework where the AI's learn from experience via TD(λ) (temporal difference) reinforcement learning is proposed in [8].

Other works somewhat related to GBG are: *General Video Game Playing* (GVGP, [9]) is a related field which tackles video games instead of board games. Likewise, μRTS [10],[11] is an educational framework for AI agent testing and competition in real-time strategy (RTS) games. *OpenAI Gym* [12] is a toolkit for reinforcement learning research which has also a board game environment supporting a (small) set of games.

II. METHODS

A. Introducing GBG

We define a **board game** as a game being played with a known number of players, $N = 1, 2, 3, \ldots$, usually on a game board or on a table. The game proceeds through actions (moves) of each player in turn. This differentiates board games from video or RTS games where usually each player can take an action at any point in time. Note that our definition of board games includes (trick-taking) card games (like Poker, Skat, ...) as well. Board games for GBG may be deterministic or non-deterministic.

What differentiates GBG from GGP? – GGP usually solves a tougher task, each agent is a *Tabula Rasa*, i. e. no game-specific features are known to the AI's at compile time. But then GGP is usually only concerned with rather simple games or – if it tackles more complex games like Connect-4 or Othello – it reaches only a very modest playing strength on them. We aim with GBG at a different goal: A framework where the game or AI implementer has the freedom to define features or symmetries (see Sec. II-F) at compile time which he believes to be useful for her game, but where the learning of the game tactics (when to perform which action) is completely left to the AIs (e. g. learning through self-play). Yet the features and symmetries are embedded in a generic interface, so that the agents are general and can be applied to any game. We then aim at developing AI's which learn to play perfectly on simple games and which exhibit a decent playing strength on games of larger complexity. We include in GBG both deterministic and non-deterministic games with stochastic elements, while GGP can only deal with deterministic games.

To summarize, we propose with GBG a framework which differs from existing frameworks in the following aspects:

- GBG is written in Java and thus available on most OS platforms.
- It is available as open source from GitHub[2]. It is as such well-suited for educational and research purposes.
- It allows to tackle both deterministic and non-deterministic games.
- It aims at developing agents for arbitrary N-player board games. It thus allows agents to be tested on a variety of games with different characteristics.
- It allows agent-agent competitions, human-agent play and inspection of the agents' value function for specific game states. The last two points are valuable for the educational perspective, since they allow to get a better understanding how a certain trained agent works and where it might have deficits.
- It offers – besides many other AIs – for the first time a **generic** TD(λ)-n-tuple agent (Sec. II-C). With 'generic' we mean that the n-tuples are defined for arbitrary game boards (hexagonal, rectangular or other) and that the same agent can be applied to 1-, 2-, ..., N-player games.
- It allows the user to define game specific features and symmetries (Sec. II-F) which are embedded into the agent framework in a generic way.

B. Class and Interface Overview

A detailed description of the classes and interfaces of GBG is beyond the scope of this article. All classes, the underlying concepts and class usage aspects are described in detail in a technical report elsewhere [13]. Here we give only a short overview of the most relevant classes in Tab. I.

C. Agent Overview

Table II provides an overview of the agents currently available in GBG. Further agents are planned to be included. Agents are implemented in such a way that they are applicable to N-player games with arbitrary N. Therefore we do not use the Minimax algorithm, which is only for 2-player games, but its generalization Max-N as suggested by Korf [14], which is viable for any N.

Expectimax-N is the generalization of Max-N for nondeterministic games. Its principle is exemplified in Fig. 1. If we predict the score tuples in the leaves with any other agent, then

[2] https://github.com/WolfgangKonen/GBG

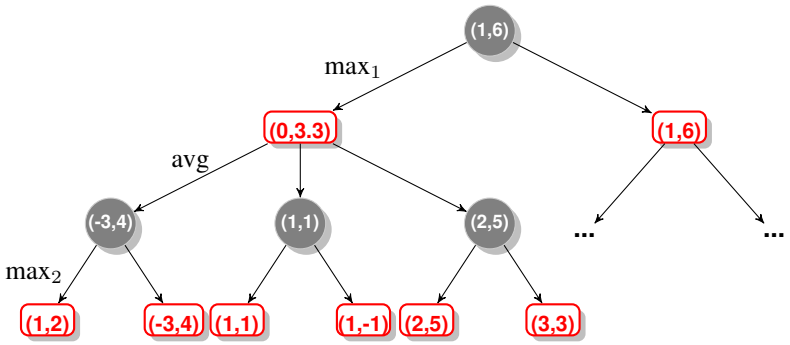


Figure 1. Expectimax-N tree for N-player games. Expectimax-N is a generalization of Max-N [14] for nondeterministic games. Shown is an example for $N = 2$ and depth $d = 3$. A node contains a tuple of game values for each player. The first level maximizes the tuple entry of the player to move (here: 1st player), the second level calculates the expectation value of all child nodes (grey circles), each having a certain probability of occurrence. The third level maximizes the tuple entry of the next player (here: 2nd player). At the leaves, the score tuple is obtained from the reward function of *StateObservation*.

Agent	Description
Max-N	'Minimax' for N player [14]
Expectimax-N	Max-N extension for nondeterministic games
MC	Monte Carlo (Sec. III-B1)
MCTS	Monte Carlo Tree Search [15]
MCTS-Expectimax	MCTS extension for nondeterministic games
TDS	TD acc. to Sutton [17] with user-supplied features
TD-n-tuple	TD agent with n-tuple features [19]
RandomAgent	completely random move decisions
HumanAgent	for human-agent or human-human play

Table II
BUILT-IN AGENTS OF GBG. THE FIRST SEVEN AGENTS ARE AI AGENTS.

Max-N or Expectimax-N of depth d becomes a generic d-ply look-ahead wrapper for that other agent. This can strengthen the quality of the other agent (at the expense of increased computational costs at play or competition time). Wrapper Max-N should be used for deterministic games, Expectimax-N for nondeterministic games.

TD-n-tuple is the coupling of TD reinforcement learning with n-tuple features. An n-tuple is a set of n board cells. If a cell has L possible states, then there are L^n n-tuple features, and each feature has a trainable weight. A network of n-tuples often has millions of weights. With reinforcement learning the network learns which of them are important.

More information on the agents in Table II is found in the relevant literature: MCTS [15], MCTS-Expectimax [16], TDS [17], [18], TD-n-tuple [19], [20], [21], [22].

D. Competitions

A game **competition** is a contest between agents. Multiple objectives play a role in such a contest: (a) the ability to win a game episode or a set of game episodes, maybe from different start positions or against different opponents (results clearly depend on the nature of the opponents); (b) ability to win in several games; (c) time needed during game play (either time per move or budget per episode or per tournament); (d) time needed for setting up the agent (training). Other objectives may play a role in competitions as well.

Competition objectives may be combined, i. e. how is the agent's ability to win if there is a constraint on the play time budget. When running this with different budgets, a curve 'win-rate vs. time budget' can be obtained. – Methods from multi-objective optimization (e. g. Pareto dominance) can be used to compare and visualize competition results.

GBG currently supports pairwise, multi-episode competitions between agents. For the future it is planned to include multi-agent competitions (round-robin tournaments) and to include time aspects in competitions.

E. Game Overview

The following games are currently available in GBG:

- Tic-tac-toe, a very simple 2-player game on a 3x3-board, mainly for test purposes,
- 2048 [23], a 1-player game on a 4x4-board with a large state space,
- Hex, a scalable 2-player game on a diamond-shaped rectangular grid. It can be played on all sizes from 2x2, where it is trivial, up to arbitrary sizes, with 11x11 being a common size. 11x11 Hex has more legal states than Chess and a high branching factor.

For the near future it is planned to port our existing game framework on Connect-4 [20], [21] to GBG as well. Other games, especially those for 3 players and more, will follow.

F. Symmetries

Many board games exhibit symmetries, that is transformations of board states into equivalent board states with identical game value. For example, Tic-tac-toe has 8 symmetries (4 rotations $\times$ 2 mirror reflections). Game learning usually proceeds much faster, if symmetries are taken into account. GBG offers a generic interface to code symmetries and to use them during training.

III. RESULTS

A. The Educational Perspective

Here we report on the first educational progress made with the GBG framework. The initial beta version of this framework was released at the beginning of 2017. Two computer science students were interested in doing their Bachelor theses in this area [16], [24]. The first student starting in January'17

Agent	Average Score
MCTS	34.700 ± 4.100
MC	51.500 ± 6.300
MCTSE	57.000 ± 6.400
TD-n-tuple, 1 ply	108.000 ± 7.600
TD-n-tuple, 5 ply	182.000 ± 6.400

Table III
RESULTS WITH GBG ON THE GAME 2048 [16]. SCORES ARE AVERAGED OVER 50 EVALUATION GAMES. THE RESULTS WITH TD-N-TUPLE WERE OBTAINED AFTER 200.000 TRAINING GAMES. THE 5-PLY RESULTS ARE ACHIEVED BY WRAPPING THE TD-N-TUPLE AGENT IN AN EXPECTIMAX-N AGENT OF DEPTH 5.

brought up the idea to solve the game 2048 with AI techniques (genetic algorithms, MCTS or similar). The second student started several months later and he was interested in tackling the scalable game Hex (the board size can be varied from 2x2 to 11x11 or even larger). Both worked enthusiastically on the topic and could generate first results within days or weeks.

This would not have been possible without the GBG framework since developing and debugging a TD-n-tuple or MCTS agent is normally out of reach for a 6-12 weeks Bachelor thesis. Thanks to the framework, both (and other) agents were available and could be readily used for research and competitions. Similarly, the presence of a standardizing interface and the availability of sample agents made it easier for the students to develop variants and enhancements. In the case of 2048, new agents MC and MCTS-Expectimax were developed (Sec. III-B1).

Finally, the resulting code is much better re-usable than code from individual projects since it is structured around well-defined interfaces. At the same time, using the interfaces for different games clarified some drawbacks in the initial beta-version design of GBG and led to improvements in interfaces and agents.

B. First Research Results

We show in Fig. 2 some examples from the gameboard GUIs. The game learning for these two non-trivial games is only a first step, but it has delivered already some valuable insights.

1) 2048: The simple 1-player game 2048 [23] is not easy to learn for computers, since a game episode can be rather long (several thousand moves). Initially, the game learning research started with two agents: MC and MCTS. The MC agent (repeated random rollouts for each action in a certain state) was meant as a simple baseline agent, while MCTS due to its tree structure was expected to act much better. But the comparison of these two agents led to two surprising results:

- Both agents were not certain in the actions they advised: Repeating the rollouts on the same state often resulted in different actions suggested.
- MCTS was not superior to MC but instead inferior.

The first result comes from the nondeterministic nature of the game and triggered some research in the direction of agent ensembles. The reason for the second result is the nondeterministic nature of 2048 as well: If a plain MCTS is used, all next states resulting from a certain state-action pair (which differ by a random tile) are subsumed in *one* node of MCTS. This node will then carry only one specific state, all further simulations will start from that state, and this leads to a severely limited exploration of the game tree.[3] Kutsch [16] developed in response to this problem an MCTS-Expectimax (MCTSE) agent, where the tree alternates between maximizing and expectation levels, similar to Fig. 1.[4] The results in Tab. III show that MCTSE is twice as good as MCTS and slightly better than MC.

First tests with the TD-n-tuple agent at the time of the Bachelor thesis were disappointing (scores below 30.000). The reason was that the implementation was not well-suited for nondeterministic 1-player games. This triggered a redesign of TD-n-tuple along the lines of Jaskowski [22] (afterstates, new TD(λ)-mechanism for long episodes). The new version led to much better results, e. g. a score of 108.000 after 200.000 training games (6.6e8 learning actions), which is similar to [22] after 6.6e8 learning actions with his plain TD(0.5) agent. Although [22], the current state-of-the-art for 2048, reports in the end largely better final scores up to 609.000, this is only achieved by additional methods designed specifically for 2048. Here we do not want to go in this direction, but are interested in the performance of a *general purpose* TD-n-tuple agent. The comparison is only meant to be a correctness check that our implementation is comparable to [22].

We note in passing that the relative increase in performance by 68% seen in Tab. III when going from 1-ply to 5-ply (from score 108.000 to 182.000) is exactly the same as the 68%-increase in performance (from score 324.000 to 545.000) obtained in [22, Tab. V].

2) Hex: Hex is a scalable 2-player game played on a board of variable size with hexagonal tiles and diamond shape. The goal for each player is to connect 'their' opposite sides of the board (see the black and white rims in Fig. 2).

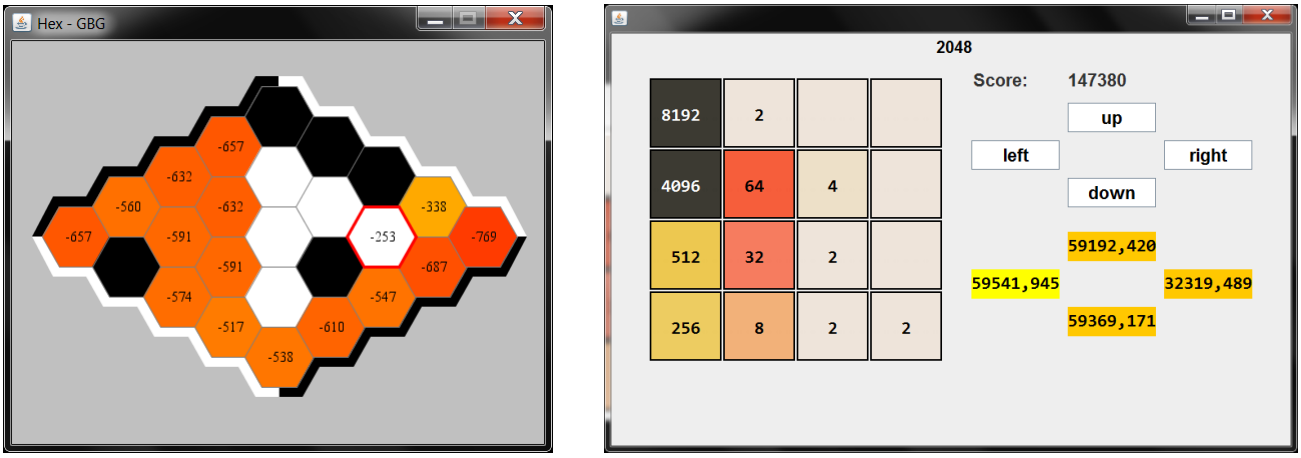
The Bachelor thesis [24] conducted on Hex in the GBG framework was to the best of our knowledge the first application of a TD-n-tuple agent to the game of Hex. Other agents were tested as well. The main results are:

- A general-purpose MCTS performs well for board sizes up to 5x5, but does not perform well on larger board sizes.
- A TDS agent with hand-designed features was successful for very small boards, but unsuccessful for all medium-size and larger boards (4x4 and up).
- A TD-n-tuple agent with random n-tuples (no game knowledge required) was successful for *all* board sizes from 2x2 up to 7x7. It could beat[5] the strong-playing computer program Hexy [25]. For board sizes 8x8 and

[3]MC on the other hand does not have this problem, since it does not store nodes.

[4]We note in passing, that Expectimax-N was not yet present at the time of the Bachelor thesis.

[5]We define *'beat'* as winning from a starting position known to be a win for that agent.



(a) Hex gameboard

(b) 2048 gameboard

Figure 2. (a) Hex gameboard example: The numbers and the color coding in the cells shows the agent's game values for the last move decision (White's turn). (b) 2048 gameboard example: The numbers in the rectangles on the right show the agent's game values for the last move decision. The yellow rectangle indicates the direction of the last move which was 'left'. Note that 'right' has a drastically lower game value than all three other choices (it would destroy the column with the high tiles).

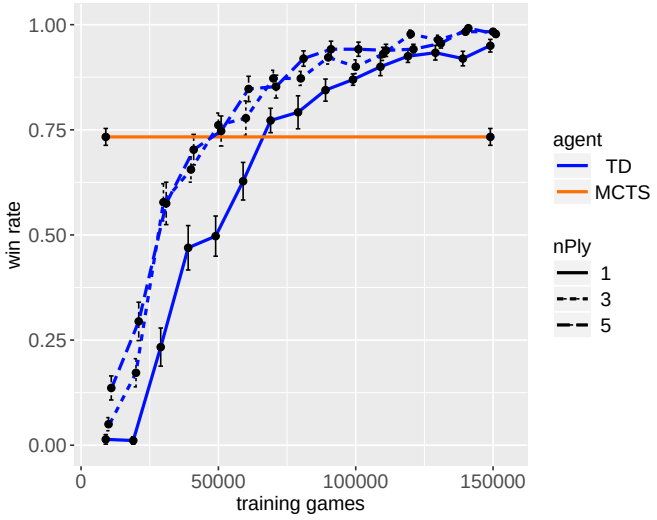


Figure 3. Training curves for TD-n-tuple agent with 25 random 6-tuples on 5x5 Hex for various d-ply look-ahead wrappers. Shown as a function of training games is the winning rate against MCTS for various starting boards where the agent can win (best is 1.0, average over 20 runs). MCTS itself (horizontal line) reaches only a winning rate of 0.74 on the same starting boards (average over 5 runs).

higher it was so far not possible to construct a TD-n-tuple agent which would beat Hexy in a competition.

The first result may come as a surprise, since it is well known that MoHex [26], a Hex playing agent based on MCTS, is a very strong playing Hex program that won several Hex Olympiads. But MoHex incorporates many enhancements specific to Hex which are (not yet) generalizable to arbitrary board games. Our MCTS is a plain *general-purpose* MCTS agent with no game-specific enhancements, and this has problems for larger Hex board sizes.

The last result is interesting from the generalization perspective: The TD-n-tuple agent, which was made popular by Lucas [19] with his success on Othello, later extended to TD(λ) by Thill [20] and successfully applied to Connect-4 [21], was taken nearly unmodified in the GBG-framework and applied to Hex without incorporating any game-specific knowledge. This demonstrates that TD-n-tuple is an approach which nicely generalizes to other games. It is much easier to apply to new games than TDS with its need for user-defined features.

Fig. 3 shows the training performance of a TD-n-tuple agent on 5x5 Hex. There is a small but significant improvement when wrapping the plain TD agent (1-ply) in a 3-ply look-ahead with Max-N. Going from 3-ply to 5-ply look-ahead gives no further improvement. TD-n-tuple outperforms MCTS.

IV. Conclusion

We presented with GBG a new framework for general board game playing and learning. The motivation for this framework came initially mainly from the educational perspective: to facilitate for students starting in the area of game learning the first steps with sophisticated agents and to facilitate competition and comparison between agents and over games.

We reported on first results obtained by students using GBG, which are encouraging: The students were able to integrate sophisticated agents into their game research and they could generate new research results within the short time span of their thesis projects. One student could contribute new agent variants (MC and MCTSE) to the GBG framework. Since it is possible to play the games and visualize results in various forms, it is easier to gain insights on what the agents have learned and where they have deficiencies.

Some of the results are also interesting from the research perspective:

- To be successful with nondeterministic games (like 2048) it is important to have appropriate nondeterministic struc-

tures in the agents as well: This is the Expectimax layer in MCTSE and the afterstate mechanism [22] in the TD-n-tuple agents.
- The general-purpose TD-n-tuple agent is successful in 2048 and in the scalable game Hex for various board sizes from 2x2 to 7x7.
- It is still an open research question how to advise the best possible n-tuple structure for larger Hex boards and whether it is possible to train such agents for 8x8 and larger boards solely by self-play.

The aim of GBG is not to provide world-championship AI agents for highly complex games like Go or Chess. This will be probably the realm of sophisticated deep learning approaches like AlphaGo Zero [1] or similar. The aim of GBG is to study agents and their interaction on a variety of games with medium complexity. Given this game complexity, results can be obtained reasonably fast on cheap hardware. The agents and their algorithms are open, easily accessible and easily modifiable by students and researchers in the field of game learning. Yet the variety of games is complex enough to make the challenge for the agents far from being trivial.

It is the hope that GBG framework helps to attract students to the fascinating area of game learning and that it helps researchers in game learning to quickly test their new ideas or to examine how well their AI agents generalize on a large variety of board games.

A. Future Work

The results presented in this paper are only a first step with the GBG framework. More games need to be implemented to assess the real generalization capabilities of agents. More agents and more elements to aid agent competition (Sec. II-D) are needed as well.

A special focus will be set on contributing a variety of N-player games with $N > 2$. Many agents available so far have been tested only on 1- or 2-player games. The extensions of TDS- and TD-n-tuple agents to $N > 2$ have been made in principle, there are however a number of options how to extend a 2-player agent to $N > 2$, which need to be tested and will then be reported elsewhere.

Acknowledgment

The author would like to thank Johannes Kutsch and Kevin Galitzki for their contributions to the GBG framework.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. 1, 6

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017. 1

[3] S. L. Epstein, "Learning to play expertly: A tutorial on Hoyle," *Machines that learn to play games*, pp. 153–178, 2001. 1

[4] M. Genesereth and M. Thielscher, "General game playing," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 2, pp. 1–229, 2014. 1

[5] J. Mańdziuk and M. Świechowski, "Generic heuristic approach to general game playing," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2012, pp. 649–660. 1, 2

[6] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the AAAI competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005. 1

[7] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008. 2

[8] D. Michulke and M. Thielscher, "Neural networks for state evaluation in general game playing," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 95–110. 2

[9] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Tech. Rep., 2013. 2

[10] S. Ontanón and M. Buro, "Adversarial hierarchical-task network planning for complex real-time games," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 2

[11] N. A. Barriga, M. Stanescu, and M. Buro, "Combining strategic learning and tactical search in real-time strategy games," *arXiv preprint arXiv:1709.03480*, 2017. 2

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016. 2

[13] W. Konen, "The GBG class interface tutorial: General board game playing and learning," Research Center CIOP (Computational Intelligence, Optimization and Data Mining), TH Köln – University of Applied Sciences, Tech. Rep., 2017. [Online]. Available: http://www.gm.fh-koeln.de/ciopwebpub/Kone17a.d/TR-GBG.pdf 2

[14] R. E. Korf, "Multi-player alpha-beta pruning," *Artificial Intelligence*, vol. 48, no. 1, pp. 99–111, 1991. 2, 3

[15] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. 3

[16] J. Kutsch, "KI-Agenten für das Spiel 2048: Untersuchung von Lernalgorithmen für nichtdeterministische Spiele," 2017, Bachelor thesis, TH Köln – University of Applied Sciences. [Online]. Available: http://www.gm.fh-koeln.de/ciopwebpub/Kutsch17.d/Kutsch17.pdf 3, 4

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998. 3

[18] W. Konen, "Reinforcement learning for board games: The temporal difference algorithm," Research Center CIOP (Computational Intelligence, Optimization and Data Mining), TH Köln – University of Applied Sciences, Tech. Rep., 2015. [Online]. Available: http://www.gm.fh-koeln.de/ciopwebpub/Kone15c.d/TR-TDgame_EN.pdf 3

[19] S. M. Lucas, "Learning to play Othello with n-tuple systems," *Australian Journal of Intelligent Information Processing*, vol. 4, pp. 1–20, 2008. 3, 5

[20] M. Thill, S. Bagheri, P. Koch, and W. Konen, "Temporal difference learning with eligibility traces for the game Connect-4," in *CIG'2014, International Conference on Computational Intelligence in Games, Dortmund*, M. Preuss and G. Rudolph, Eds., 2014. 3, 5

[21] S. Bagheri, M. Thill, P. Koch, and W. Konen, "Online adaptable learning rates for the game Connect-4," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 1, pp. 33–42, 2015. 3, 5

[22] W. Jaskowski, "Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017. 3, 4, 6

[23] G. Cirulli, 2014. [Online]. Available: http://gabrielecirulli.github.io/2048/ 3, 4

[24] K. Galitzki, "Selbstlernende Agenten für das skalierbare Spiel Hex: Untersuchung verschiedener KI-Verfahren im GBG-Framework," 2017, Bachelor thesis, TH Köln – University of Applied Sciences. [Online]. Available: http://www.gm.fh-koeln.de/ciopwebpub/Galitzki17.d/Galitz17.pdf 3, 4

[25] V. V. Anshelevich, "A hierarchical approach to computer Hex," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 101–120, 2002. 4

[26] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010. 5