A Game-centric Approach to Teaching Artificial Intelligence

Marc Pouly[a] and Thomas Koller[b] and Ruedi Arnold[c]
School of Information Technology, Lucerne University of Applied Sciences and Arts, Switzerland

Keywords: Teaching Artificial Intelligence, Card Game Playing Platform, Open Source Software, Student Competition, Contextualized Education.

Abstract: Man vs. machine competitions have always been attracting much public attention and the famous defeats of human champions in chess, Jeopardy!, Go or poker undoubtedly mark important milestones in the history of artificial intelligence. In this article we reflect on our experiences with a game-centric approach to teaching artificial intelligence that follows the historical development of algorithms by popping the hood of these champion bots. Moreover, we made available a server infrastructure for playing card games in perfect information and imperfect information playing mode, where students can evaluate their implementations of increasingly sophisticated game-playing algorithms in weekly online competitions, i.e. from rule-based systems to exhaustive and heuristic search in game trees to deep learning enhanced Monte Carlo methods and reinforcement learning completely freed of human domain knowledge. The evaluation of this particular course setting revealed enthusiastic feedback not only from students but also from the university authority. What started as an experiment became part of the standard computer science curriculum after just one implementation.

1 INTRODUCTION

The online news platform *New Atlas* coined 2017 as

The year AI beat us at all our own games

referring to the memorable victory of DeepMind's AlphaGo Master against Ke Jie, the world No. 1 ranked Go player in May 2017 and the seminal announcement of AlphaGo Zero in October 2017, stronger than any previous human champion defeating version of AlphaGo. But the same year has also seen DeepStack and Libratus independently conquering several human poker champions in No Limit Texas Hold'em as well as the participation of OpenAI in a giant eSports tournament in August 2017 beating world top players in the online battle game Dota 2 (Haridy, 2017).

Such public exhibitions of human against artificial intelligence in various strategic board, card and video games have always been attracting much interest from one of the earliest public clashes of man and machine at the industrial fair in Berlin in 1951, where an archaic computer called Nimrod beat the former secretary of commerce of West Germany in the game of Nim (Redheffer, 1948; Baker, 2010), over the legendary victories of IBM Deep Blue against the chess grandmaster Garry Kasparov in 1997 (Campbell et al., 2002) and IBM Watson against two human Jeopardy! champions in a series of public TV shows in 2011 (Kelly and Hamm, 2013), to the most recent breakthroughs in Go, poker and various eSports disciplines. Irrespective of the certainly welcome news headlines for artificial intelligence research, set off each time the human race had to surrender in another popular game, the development of intelligent game-playing agents has always had a much more significant role. After all, chess was famously referred to as *the drosophila of artificial intelligence* (McCarthy, 1990), and after 1997, this role was attached to the Chinese board game Go as experts had thought it could take another hundred years before an AI system beats a human Go champion (Johnson, 1997). Throughout the course of history, AI has mastered increasing levels of game complexities including the vast search spaces of Go, imperfect information and randomness in card games and even socio-behavioral aspects (e.g. bluffing in poker games) that are typically attributed to humans. These big headlines therefore witness the algorithmic progress of artificial intelligence research, i.e. from rule-based systems to exhaustive and heuristic search in game trees to deep learning enhanced Monte Carlo methods and rein-

[a] https://orcid.org/0000-0002-9520-4799
[b] https://orcid.org/0000-0003-2309-5359
[c] https://orcid.org/0000-0001-7185-305X

Pouly, M., Koller, T. and Arnold, R.
A Game-centric Approach to Teaching Artificial Intelligence.
DOI: 10.5220/0007745203980404
In *Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019)*, pages 398-404
ISBN: 978-989-758-367-4

forcement learning freed of any human domain expertise. At the same time, they were made possible only by important technological advances: the Nimrod was basically an public demonstration of early computer technology; Deep Blue a massively parallel architecture of dedicated chess chips (Campbell et al., 2002) whereas with Watson IBM ultimately proved their expertise in building supercomputers culminating in Google's demonstration of GPU technology for deep learning with AlphaGo and AlphaGo Zero.

This historical evolution of ideas, methods and technology complemented with all the intriguing aspects and stories around a man-machine competition and broad coverage in research literature and popular media made us raise the idea of experimentally designing an introductory class to artificial intelligence on Bachelor's level along these developments, leaving the trodden paths of just following the well-established syllabus of one of the major textbooks on the topic. Moreover, by making available a self-developed software framework together with a server application for in-class tournaments, we gave our students the opportunity not only to design and implement their own game-playing agents with increasingly complex and stronger methods, but also to frequently run tournaments and obtain direct feedback on their approaches. AI courses with in-class tournaments or even participation in open challenges, e.g. on Kaggle[1], became fairly common in recent times. However, we aimed to push this idea to an extreme by running weekly student competitions for each game-playing method presented in class. The subsequent chapters survey our first attempts and experiences in setting up this program, initially limited to a single semester course but with a clear ambition to spread over several courses with different emphasis. We will further justify our selection of a card game for these competitions and sketch the software framework and server infrastructure that we make available our code as open-source projects to the community in the hope of spreading and challenging our guiding principles and the proposed didactical method.

2 A GAME-CENTRIC SYLLABUS

Our course started off by an introduction to sequential games with perfect information laying out basic concepts such as game trees, backward induction, sub-game perfect Nash equilibria and the notion of strongly and weakly solved games. This already provides sufficient information to pop the hood of the Nimrod, maybe the first artificial intelligence to beat a human in a public exhibition in 1951, respectively to implement perfect play for a selection of simple games including Nim and tic-tac-toe. From there we intensified the discussion on state space complexity and game tree modelling, introduced the notion of zero-sum games, the famous minimax algorithm with its extension to multiple players and alpha-beta pruning. Such a brute-force analysis of the connect four game yielding a perfect player based on simple database lookup became possible in 1995. Note however that the game was weakly solved already in 1988 (Allen, 2010). We next introduced the idea of combining minimax with domain-specific heuristics and elaborated on formal requirements, manual and semi-automated design approaches to heuristics following (Russell and Norvig, 2010). This proved sufficient for a closer analysis of DeepBlue - a massively parallelized minimax implementation with alpha-beta pruning and a heuristics based on 8000 chess features manually tailored by a team of expert chess players that finally beat a chess grandmaster in 1997 (Campbell et al., 2002). This marked an unprecedented success in AI history but at the same time stands for the extreme edge of what can be achieved by a rule-based vaccination of algorithms with human domain knowledge. Also, these approaches do not carry over to games with imperfect information. Our course advanced by introducing Monte Carlo tree search (MCTS) with random play-outs and the upper confidence bound for balancing exploration and exploitation. First proposed in 2006 (Coulom, 2006), MCTS since then has completely dominated the computer Go scene (Couëtoux et al., 2013). The picture was rounded out by its variants for imperfect information games called determinization and information set MCTS (Cowling et al., 2012). We paused the historical survey at this point for an introduction to the basic principles of supervised machine learning including data preparation techniques and a small selection of classifiers adding in logistic regression, decision trees and random forests. Hereby, is is important to say that we put the main emphasis not on the algorithmic aspects of machine learning but rather on how to correctly evaluate a machine learning classifier with hyperparameter optimization. Next, an introduction to deep learning and the open source neural network library Keras[2] was given, which allowed us to resume our survey of approaches to game-playing agents by combining MCTS with value and policy functions obtained from supervised deep learning analogously to the original version of AlphaGo (Silver et al., 2016). The final part of this course comprised an introduction

[1] https://www.kaggle.com/competitions

[2] https://keras.io/

to deep reinforcement learning aligned with the approaches implemented in AlphaGo Zero (Silver et al., 2017; Silver et al., 2018). Other popular games such as backgammon, checkers, bridge or poker also deserved special mention of course.

3 SELECTION OF A CARD GAME

With every new approach to game-playing agents presented in class, i.e. from a rule-based implementation of strategies to exhaustive and heuristic search to Monte Carlo tree search and its enhancements with supervised deep learning to reinforcement learning, we tasked students to implement these techniques and evaluate within the scope of a series of in-class tournaments. This imposes very concrete constraints on the sort of game to be selected, as we also wanted to always use the same game in order to compare different approaches and also generate hybrid techniques throughout the course. For the following reasons we thought that a card game would constitute a good choice: Many card games can be instantiated as either two-player games or with more than two players considered as teams or individual players. We made available a dedicated (partial) cheating mode that allows each player to see the hands of its team mates. Card games ultimately are games of imperfect information. However, in a (full) cheating mode, where each player can see every other player's hand, methods limited to perfect information such as minimax with alpha-beta pruning can be applied. As an interesting side effect, this allows comparison of players with perfect and imperfect information, e.g. pure Monte Carlo tree search (cheater) vs. information set Monte Carlo tree search (honest player) in order to quantitatively assess the advantage of perfect information. Card games often have a fixed number of tricks played, while the number of moves in chess matches may differ significantly. This simplifies many aspects from runtime scheduling of in-class tournaments to generation of training data, etc. Also, when tournaments are to be run on a weekly basis, they must be efficient and manageable, i.e. given the number of teams and a limit for the duration of each move, then the total runtime of a tournament must be predictable. Finally, there is a seemingly uncountable number of local variants of card games played all over the world, and we hope that our game platform proves generic enough for integrating many such variants.

Our library comes along with an implementation of the well-known single-player game Hearts (without initial passing round) and a variant of the popular Swiss card game Jass. We chose the latter for our in-class tournaments because of its increased complexity: Jass requires one player to initially declare a trump mode. This affects the priority and value of cards in the follow-up game. Second, we intentionally chose a game of only regional spread. Students may find various implementations of algorithms for widespread games on the internet, and therefore rightly have enough sources for inspiration, but to our best knowledge no code repository for the card game variant we chose is publicly available. Finally, we appreciated support from an online provider of the Jass card game in Switzerland, who made available anonymized data from human plays for supervised machine learning (trump prediction and policy / value networks for MCTS). More specifically, we were given nearly 500k rounds of human play. Finally, another main reason for the selection of Jass is its ongoing popularity in Switzerland. Jass is arguably the most popular card game in our country with massive numbers of regular players, weekly shows on national TV and nation-wide tournaments. As stated in many sources, e.g. (Weinstein et al., 2018; Tapola et al., 2013; Guzdial, 2010; Arnold et al., 2007), teaching can be particularly efficient whenever students can use concrete examples and establish a connection between the subject to learn, and their own experiences in everyday life, i.e. if education is contextualized. Retrospectively, we feel confirmed in this choice and strongly encourage other educators to bring in such regional flair whenever possible.

4 FRAMEWORK AND SERVER

We wanted students to focus on the programming of their solutions immediately at the start of the first exercise and thus provided a framework with the core functionality to program a player. This *jass-kit* includes:

- Data classes which contain the complete information of the state of a game from a player's point of view including all cards that have been played, the trump mode selected and the player's hand. A separate class containing the complete information of the game including the hands of team mates or even all players can be used to implement cheating version (perfect information game) for comparison of algorithms. Of course, this feature is not available in tournament mode.
- Implementation of the rules for the games Hearts and Jass (*Schieber* variant). This includes an implementation of the logic of the game, for example when a card is played as the last card of a trick, the

winner is determined, the score is updated and the player for the next card is determined. Also, the valid cards to play can be determined in a configuration facilitating the implementation of player algorithms.

- Functionality to parse and convert the available log files for supervised learning.
- A player interface that can be augmented with a specific implementation of a player's algorithm. The player can be tested using an implementation of a so-called *Arena* class for playing against other local players or can be integrated into a provided web server implementation to play remotely or in a tournament.

Python 3.5[3] was used for the implementation of the framework augmented with numpy[4] for arrays, matrices and mathematical methods. The students had access to the complete jass-kit framework repository, and a demo-kit repository with a sample implementation of a basic player. The tournament server business logic was implemented in Python using flask[5] for its REST interface and MongoDB[6] for preserving state. The server also tracks connection errors and timeouts when contacting the players to facilitate debugging.

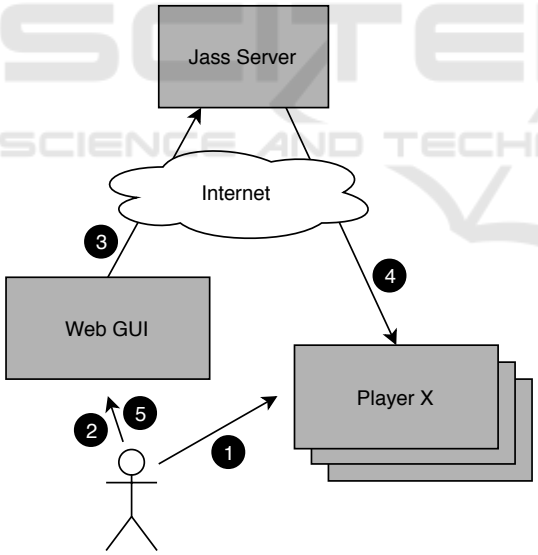

Figure 1: A high-level system overview showing the three main components *player implementations*, *Jass server* and *web GUI*, as well as a typical usage sequence from a user's perspective.

In order to allow students a simple setup and experiments with their own Jass bots, we further provide a web-based graphical user interface (GUI). On this public available website the students can easily configure and start new games and tournaments, inspect progress and download the results. See figure 1 for an overview of our system and a typical interaction sequence in five steps:

1. A student sets up her own player instance(s) on her machine(s).
2. A student creates a new game or registers her player(s) for one of the available games and then starts it.
3. The web GUI sends all the relevant information to the jass server which then executes the desired game with the registered players.
4. During the execution of a game, the jass server sends the necessary game information to the player instances and requests the desired actions.
5. Once a game is finished, students can see the results on the GUI and download a file with all details about the game played. This information is of course persisted and provided by the jass server.

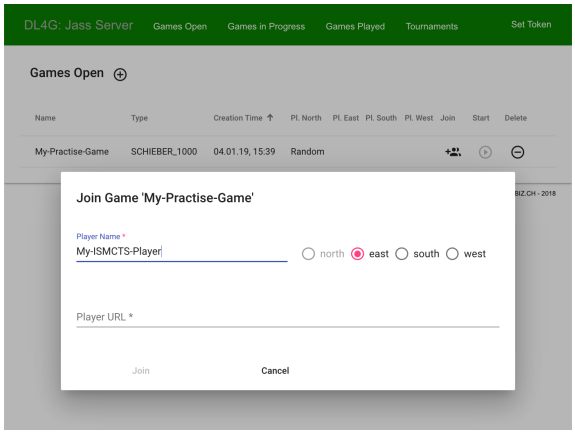

Figure 2: Sample screenshot of the web GUI showing the registration of a player.

Figure 2 displays the registration screen for a new player to an open game. This Angular-based[7] application was available for the students throughout the semester and still is on our website[8]. For this course we developed the following four projects:

- **jass-kit:** base components for the games of Jass and Hearts including e.g. data classes, game rules logic, a game arena, and so on. The code allows for easy integration of other card games.

[3] A later version of Python was not available, as compatibility with the tensorflow framework required version 3.5.
[4] http://www.numpy.org
[5] http://flask.pocoo.org
[6] https://www.mongodb.com
[7] https://angular.io
[8] https://jass-server.abiz.ch

- **jass-demo:** a sample player for students to allow for a quick start;
- **jass-server:** a server for the setup, execution and persistence of games and tournaments using web-based players;
- **jass-server-web-gui:** a GUI for our server.

As we would like to encourage other lecturers to experiment with this teaching approach, we make our source code available to the community under an Apache 2.0 license. The above mentioned projects are therefore publicly available in our code repositories[9].

5 FINAL TOURNAMENT

At the end of the semester, a tournament was carried out between the different student teams. Each team was allowed to compete with one or more player implementations. The rules of the tournament were taken from the rules commonly used in the Swiss Jass championships where multiple sets of 12 games are played and the total number of points won are counted. Each set is played against a different opponent. The implemented pairing algorithm uses a greedy algorithm to sequentially assign random opponents not yet paired and not previously encountered, if possible. A total of 13 player implementations were registered from the students and augmented with one of our implementations to add up to an even number of players.

Additionally a voluntary challenge tournament was set up, where the students could challenge implementations from the lecturer team.

Throughout the semester we presented the various game playing approaches to the students as depicted above, but no constraints were imposed on the technique to be implemented for the final competition. The majority of the teams used a deep learning approach for both trump and card selection. Many filtered the input data by calculating statistics of human players from the provided log files and only chose frequent players for training or even only players that performed better than average. A minimal implementation of input features was already given during the lecture and about half the teams expanded these to include more data. Players that filtered training data and trained for several hours instead of only minutes generally got better results. While many teams additionally implemented MCTS, only two used it for the tournament with one implementation notably reaching place 5 in the tournament. The winning team was the only one to employ a deep convolutional neural network. None of the teams was able to realize a combined MCTS and deep learning approach that played well enough even though several tried, but ended up competing in the tournament with another approach. Figure 3 shows a screenshot of the final tournaments' ranking.

Our own bot that was included in the tournament as a filler was an implementation of a deep neural network using more features including information which card was played by which player and trained on a non-filtered data set. About half of the teams were able to beat this implementation both with other deep neural networks and MCTS implementations.

Tournament Players

Name	Score ↓	Games Played	Games Won	Games Lost	Games Tied
Joseph	9186	8	7	1	0
Inceptotronic	8671	8	8	0	0
bothiger	8525	8	4	4	0
ArtificialCardHoppers	8439	8	6	2	0
C-3PO	8047	8	6	2	0
DeepDonald	7999	8	5	3	0
ABIZ DNN (Filler)	7776	8	5	3	0

Figure 3: Screenshot showing the top ranked players of the final tournament.

While AlphaGo Zero (Silver et al., 2018) notably only uses several hundert iterations for their MCTS-based reinforcement learning algorithm, pure MCTS methods tend to need several thousands or ten thousands iterations at least to perform well. This is difficult to achieve in a Python implementation for a reasonable computing time. We therefore implemented a C++ version of our framework with the intension to use it in future lectures and tested it using two MCTS-based players in the voluntary challenge tournament. The C++ framework was also made available to students towards the end of the semester, but only one team tried to employ it. 12 student teams participated in the challenge tournament, but none was able to win against our implementation, which used a DNN network to select trump and a MCTS implementation with 250′000 iterations to select the played card. The third place, with only losing one game, was occupied by the same team that won the other tournament.

[9]https://gitlab.enterpriselab.ch/jass/info/

6 EVALUATION

There was no thorough evaluation of this module and we doubt that a statistically expressive evaluation of this teaching method will ever be possible for us. At least at our university, the student count for such a non-mandatory module will always be rather low with high bias as we naturally attract only students interested in AI and machine learning. Finally, only a few students tend to participate in course evaluations. However, the computer science department authority charged an external provider with an evaluation, from which we received 15 qualitative feedbacks out of the 29 students registered to this course. The figures rank this module among the top-5 percent of modules taught in the computer science department. However, as mentioned above, please take these figures with a grain of salt. From the textual feedbacks we could extract great support for this teaching method, the tremendous effort from the lecturer team was highly appreciated, and many students submitted ideas and feature requests for our game platform. On the negative side, some students associated an over-average workload with this module and an increased entry barrier due to the extensive software frameworks made available. We are confident that we can address the latter issue by an additional tutorial session in the first semester week. Also, we plan to elaborate more direct feedback mechanisms for students: Through participation in tournaments students learn about the performance of their player implementations relative to other student teams. By additionally including a player implementation of normed strength (e.g. a random card player in the simplest case) we could impose a lower bound on the average performance each student team should be able to achieve.

As a consequence of the enthusiastic feedback mentioned above, the authority of our department decided to instantly integrate this course into the standard computer science curriculum.

On a personal note, despite of our long-term experience as researchers and lecturers of artificial intelligence classes, we have never encountered such an active discussion among students on what *intelligence* exactly means. Indeed, throughout this course students have encountered and implemented very different conceptions of artificial intelligence, from manually programmed rules to brute-force search to injected human domain knowledge over extracted patterns from human data to finally an intelligent computer player that, apart from the elementary game rules, does not contain any piece of human knowledge anymore. In that respect we can affirm the use of games as a motivation tool in computer science curricula as for example presented in (Bezakova et al., 2013) along with many references to other game-based approaches to teaching computer science.

7 CONCLUSION AND OUTLOOK

This article exposes our ideas and first experiences of designing an introductory class to artificial intelligence along the historical development of game-playing agents. As the centerpiece of this innovation we made available a self-developed software framework, server infrastructure and web-based graphical user interface for students to implement increasingly sophisticated game-playing algorithms and to evaluate their code in weekly online competitions, which is a substantially different approach to just carrying out a single tournament at the end of an implementation project that usually spans over several semester weeks. We intentionally chose a card game for these regular competitions in order to bring in the additional challenges of cooperation and imperfect information. Furthermore, there is a huge variety of regional card games all over the world, for which students rightly find inspiration in online resources but no ready-made solutions. This course setting naturally encouraged students to let game-playing algorithms of different degrees of sophistication compete against each other (e.g. how does a simple rule-based agent perform against deep learning augmented Monte Carlo tree search), to try out hybrid solutions or to quantitatively assess the advantage of perfect information (cheating) by letting e.g. Monte Carlo tree search compete against its extension to information sets. We received enthusiastic feedback from students and university authority, which motivated us to share these ideas and make our infrastructure available as an open-source project to other lecturers.

We are looking forward to gaining more experience and insights employing our approach to teaching artificial intelligence and to improve our course. And we hope this game-based approach will inspire fellow lecturers. In the Los Angeles Times, Murray Campbell (Lien and Borowiec, 2016) called AlphaGo's victory *[...] the end of an era [...] board games are more or less done and it's time to move on*. – Agreed, it is time to move on to teaching and making these methods accessible to the next generation of computer scientists.

ACKNOWLEDGEMENTS

We thank our students for their great engagement and valuable feedback as well as our department authority for the unconditional support of this didactical experiment and the motivating decision that this module will become regular part of the computer science curriculum at HSLU. We further greatly appreciated the short-term provision of prizes for the winning student teams by Bison Schweiz AG. We finally are much obliged to Roland Christen for supporting development and deployment.

REFERENCES

Allen, J. (2010). *The Complete Book of Connect 4: History, Strategy, Puzzles*. Sterling Publishing Company, Incorporated.

Arnold, R., Langheinrich, M., and Hartmann, W. (2007). Infotraffic – teaching important concepts of computer science and math through real-world examples. In *Proceedings of the 38th ACM SIGCSE Technical Symposium*, pages 105–109, New York. ACM Press.

Baker, C. (2010). Nimrod, the world's first gaming computer.

Bezakova, I., Heliotis, J. E., and Strout, S. P. (2013). Board game strategies in introductory computer science. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 17–22, New York, NY, USA. ACM Press.

Campbell, M., Hoane, A., and hsiung Hsu, F. (2002). Deep blue. *Artificial Intelligence*, 134(1):57 – 83.

Couëtoux, A., Müller, M., and Teytaud, O. (2013). Monte carlo tree search in go.

Coulom, R. (2006). Efficient selectivity and backup operators in monte carlo tree search. In *Proceedings Computers and Games 2006*. Springer-Verlag.

Cowling, P. I., Powley, E. J., and Whitehouse, D. (2012). Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143.

Guzdial, M. (2010). Does contextualized computing education help? *ACM Inroads*, 1(4):4–6.

Haridy, R. H. R. (2017). 2017: the year ai beat us at all our own games.

Johnson, G. (1997). To test a powerful computer, play an ancient game.

Kelly, J. E. and Hamm, S. (2013). *Smart Machines: IBM's Watson and the Era of Cognitive Computing*. Columbia University Press, New York, NY, USA.

Lien, T. and Borowiec, S. (2016). Alphago beats human go champ in milestone for artificial intelligence.

McCarthy, J. (1990). Chess as the drosophila of ai. In Marsland, T. A. and Schaeffer, J., editors, *Computers, Chess, and Cognition*, pages 227–237, New York, NY. Springer New York.

Redheffer, R. (1948). A machine for playing the game nim. *The American Mathematical Monthly*, 55(6):343–349.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, third edition.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354 EP –.

Tapola, A., Veermans, M., and Niemivirta, M. (2013). Predictors and outcomes of situational interest during a science learning task. *Instructional Science*, 41:1–18.

Weinstein, Y., Sumeracki, M., and Caviglioli, O. (2018). *Understanding How We Learn: A Visual Guide*. Routledge, 1st edition.