Chess as Played by Artificial Intelligence

Filemon Jankuloski, Adrijan Božinovski[1]

University American College of Skopje, Skopje, Macedonia

filjankuloski@gmail.com

bozinovski@uacs.edu.mk

Abstract. In this paper, we explore the evolution of how chess machinery plays chess. The purpose of the paper is to discuss the several different algorithms and methods which are implemented into chess machinery systems to either make chess playing more efficient or increase the playing strength of said chess machinery. Not only do we look at methods and algorithms implemented into chess machinery, but we also take a look at the many accomplishments achieved by these machines over a 70 year period. We will analyze chess machines from their conception up until their absolute domination of the competitive chess scene. This includes chess machines which utilize anything from brute force approaches to deep learning and neural networks. Finally, the paper is concluded by discussing the future of chess machinery and their involvement in both the casual and competitive chess scene.

Keywords: Chess · Algorithm · Search · Artificial Intelligence

1 Introduction

Artificial intelligence has become increasingly prevalent in our current society. We can find artificial intelligence being used in many aspects of our daily lives, such as through the use of Smartphones and autonomous driving vehicles such as the Tesla. There is no clear and concise definition for Artificial Intelligence, since it is a subject which still has massive space for improvement and has only recently been making most of its significant advancements. Leading textbooks on AI, however, define it as the study of "intelligent agents", which can be represented by any device that perceives its environment and takes actions that maximizes its chances of achieving its goals [1]. In the context of this paper, these intelligent agents are chess playing AI, which implement several different types of methods and strategies for a single purpose: to defeat any opponent in a game of chess. We will take a look at how the strategies used to develop these chess playing AI engines have progressed and evolved over the span of AI's 70 year existence.

2 Overview of Impactful Chess Machinery and Programs

Table I. Chess programs and machinery

Name of program/machine	Names of creators	Year of creation	Algorithms, methods, or technology incorporated
Turochamp	Alan Turing, David Champernowne	1948	Variable lookahead, Two move heuristic, Evaluation of positions based on mobility, piece safety, king mobility, king safety, castling, and more.
NSS	Allen Newell, Herbert Simon, Cliff Shaw	1958	Move generator, Position evaluator, Alpha Beta searching
Mac Hack VI	Richard Greenblatt	1966	Move generator, Position evaluator, Alpha Beta searching, Transposition table
First Macedonian chess program	Stevo Božinovski	1969	Move generator, Position evaluator
Chess (Northwestern University Chess 4.5)	Larry Atkin, David Slate	1975	Move generator, Position evaluator, Alpha Beta searching, Transposition table, Bitboard, Full width search, Iterative deepening
Belle	Ken Thompson, Joe Condon	1976	Move generator, Position evaluator, Alpha Beta searching, Transposition table, Lazy and full evaluation, Principal variation splitting
HiTech	Hans Berliner, Carl Ebeling, Murray Campbell, Gordon Goetsch	1985	Move generator, Position evaluator, Pattern recognition, Transposition table, Parallel searching, Alpha Beta searching
Fritz	Frans Morsch, Mathias Feist	1991	Move generator, Position evaluator, Parallel searching, Null move search
Deep Blue	IBM Development Team	1996	Move generator, Position evaluator, Alpha Beta searching, Transposition table, Parallel searching, Singular extension
StockFish	Tord Romstad, Marco Costalba, Joona Kiiski, Gary Linscott	2016	Move generator, Position evaluator, Iterative deepening, Parallel Search, Transposition Table, Move valuable victim/least valuable aggressor, Null move search, Singular extensions, Futility pruning, Static exchange evaluation
AlphaZero	DeepMind Development Team	2017	Neural networks, Deep learning, Reinforcement learning, Monte Carlo Tree Search, Transposition table, Tensor processing unit

3 Chess Machinery

3.1 Turochamp

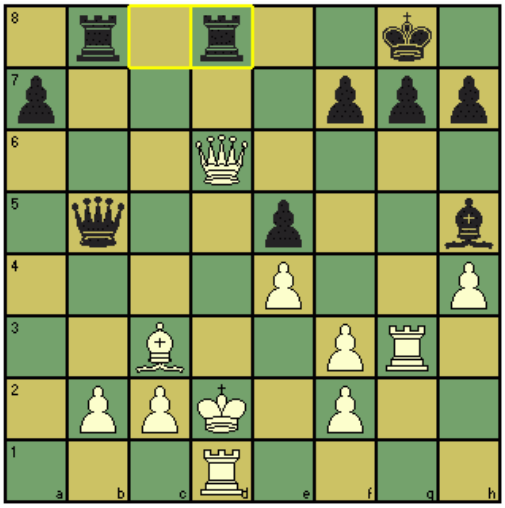

Fig.1. Turochamp's interface [2]

One of the first chess programs to ever be created was the Turochamp in 1948. Alan Turing and David Champernowne developed the program to be able to compete with players in chess at a low level using a two move heuristic [3]. Firstly, an opponent's move would need to be used as input, and the output would be calculated as a response. The calculation is done by treating each space that a piece could move to as a "state" and measuring how many points could be acquired by moving to said "state". The criteria for number of points is an extensive list which includes: the mobility of each piece, the safety of each piece, the threat of a checkmate, the value of the player's piece if it is taken, the value of a piece that may be taken by the moving piece, and much more [4]. The amount of points a piece is worth is also dependent on the piece itself. For example, a rook would be worth more than a pawn, and the queen would be worth more than both the rook and the pawn. According to Champernowne, the main objective of the program is not just to defeat its opponent, but also to decide whether or not a piece should be taken [4]. It is also worth noting that, due to the complexity of the program, it was never executed on a computer as technology was not capable of running code on that level yet. However, it was still the first computer chess program conceived that began development, and was published in 1951.

3.2 NSS

Fig. 2. NSS being run with the help of the JOHNNIAC computer [5]

In 1958, NSS became the first chess program capable of utilizing the Alpha Beta search algorithm [6]. This algorithm is used to search with a depth of 2 within the chess board. Taking the depth into consideration, we will consider all possible moves that we can make, and also take into consideration all possible counter responses that can be made to that move. Let's say we have a move called "Move A" and a move called "Move B". Move A guarantees us an even position with the opponent, while there is a scenario with Move B which results in the loss of one of our pieces. With this knowledge in mind, we can say that Move A is "better" than Move B since Move A at least gives us an even position regardless of where it's moved. Move A helps us define a lower bound, and this bound helps us determine when a move is too bad to even take into consideration.. The upper bound also needs to be defined, because if we search with a depth greater than 3 and we continuously attempt to make moves which benefit us, the opposing player simply will not let this happen as they could just avoid this sequence of advantageous moves. The Alpha and Beta in this algorithm's name comes from the lower bound and upper bound, respectively. Using the Alpha Beta algorithm, NSS became the first chess program which defeated a human opponent in 1958 [7].

3.3 Mac Hack VI

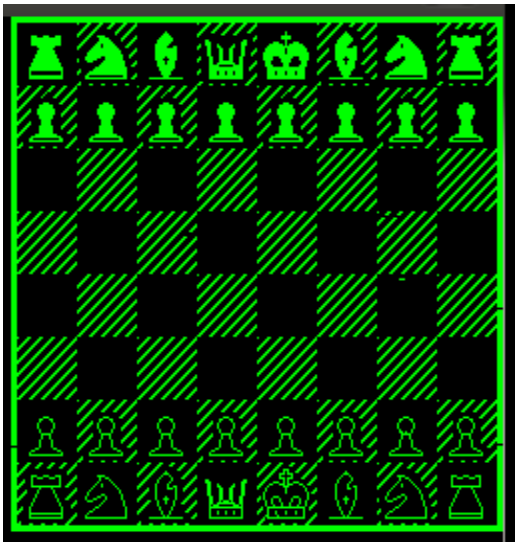

Fig. 3. Mac Hack's bitmap display [8]

The Mac Hack was fully developed in 1967 by Richard Greenblatt, and, more notably, it was the first chess program to utilize a transposition table [9]. As aforementioned, computers were incapable of running the first chess programs which were published due to the complexity of their codes. At the time of the Mac Hack's creation, there were still many computer programs using the Alpha Beta algorithm which took a tedious amount of time just to make a single move. The introduction of transposition tables offered a solution to this problem. Instead of calculating the amount of points gained from moving a chess piece to a particular state repeatedly, Mac Hack would simply hash the calculation and keep it stored in a table. In a case where it would have to analyze the same position, instead of making the same redundant calculation, it would simply locate where the calculation was stored in the table instead of wasting precious time that could be spent making new calculations. In an actual game of chess, the program would first check to see if a position had been previously analyzed before making any calculations, and, if it had been analyzed, it would simply retrieve the calculation from the table. Unfortunately, due to the lower memory capacity of computers at the time, there was a limit on how many positions could be stored inside of a table. Should a new position need to be stored inside of the table, the program would simply remove the calculation of a position which had been accessed the least. As a result of the implementation of these transposition tables, reduction of search time could range anywhere between 0 percent and a whopping 50 percent [10]. Utilizing its Alpha Beta algorithm and transposition table, Mac Hack became the first chess program to defeat a human opponent in a tournament setting in 1967 [11].

3.4 First Macedonian Chess Program

The first Macedonian chess program was created by Stevo Božinovski in 1969 and it was written in Fortran for the IBM 1130 computer [12]. The program was not meant to complete a full game, but rather simulate a scenario where the human player has a black king, and the computer has a white king and a white rook. This program used a call named "DATSW" in order to plot the chess board every time a chess piece was moved [12]. The program also consisted of 3 different subroutines. "POTEZ" was used to determine if a move made by

the human was legal or illegal, “KODER” encoded the move written in standard chess notation (KE6) into international representation, and “POZIC” served the purpose of both analyzing positions and generating moves for the program to make [12]. There was also a function named “MATIR”, which was used to determine if the current state of a chessboard was in checkmate.

3.5 Chess

Fig. 4. Chess’ 4.6 version ran on a chess machine named “Chesstor” [13]

After nearly a full decade with little to no progress in the evolution of chess programs, the Northwestern University chess program 4.5 was finally introduced in 1975 after 8 years of development [14]. The full width search and iterative deepening depth first search is what separated the Chess 4.5 program from its predecessors. The full width search, much like the name implies, is a search where all directions that a chess piece can move to are considered. In the early stages of a chess game, such a search is purely impractical due to how extensive they would be. However, in the late stages of a game where there are few pieces left, a full width search would prove to be far more useful than the basic Alpha Beta algorithm and other selective heuristics. Iterative deepening searches are much like full width searches, except that rather than only exploring how many different directions a chess piece can go, it also checks how far a particular chess piece can go in a certain direction, and whether or not there is a position to be secured that would be in favor of the program’s victory. One would think that such a search would be costly, however, in actuality, this search is quite memory efficient. When a goal node is found, or for example, when a bishop piece finds a piece to attack, then it unwinds without searching any other positions which may lie ahead of the piece that may be attacked. The Chess 4.5 program is also the first program to utilize bitboards. Basically, a bitboard is a 64 bit data structure in which there is a bit for each corresponding space on the chessboard. Each bit is labeled according to the following structure: from left to right, the columns are labeled from “a” to “h” and from bottom to top, the rows are labeled from “1” to “8”. In other words, the very bottom left space is labeled “a1” and the very top right space is labeled “h8”; a1 is the 0th bit and h8 is the 63rd bit respectively. Utilizing several different algorithms and technologies at once, the Northwestern University Chess program became the model for all future chess development and it dominated all other computer chess programs and computer chess program tournaments until the early 80s. It is also worth noting that when Chess 4.5 was updated to

Chess 4.7, it became good enough at chess to be able to win against a chess master, David Levy, in 1978 [15]. This was the first time a chess program was able to take a game off of a human chess master.

3.6 Belle

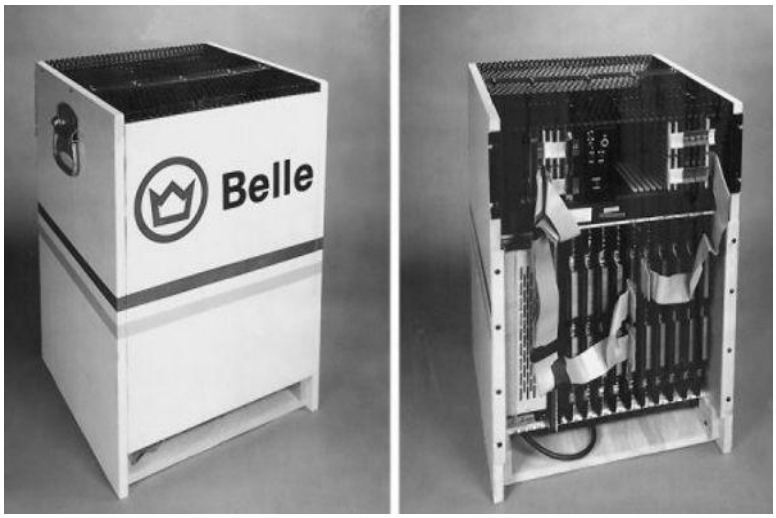

Fig. 5. The chess machine "Belle" [16]

Belle was the first chess machine that achieved a master ranking in chess in 1983 [17]. It was also the first chess machine which used specialized hardware specifically for itself in order to improve its chess play. Belle's chess program implemented a move generator, which was updated from 1976 up to 1980, from the first generation up to the third generation. In the first generation, a 6 bit "from" register searches the board for friendly chess pieces. When a friendly piece is found, an offset bit code is generated. The code generated from the starting position of the chess piece is aggregated to the code generated from the friendly piece's movement, and this resulting code describes a potential move. This is the process for how a single position is generated, stored, and analyzed all at once. In the second generation, the generator had implemented several improvements such as a stack in which it could store its moves, a position evaluator, and a transposition table to memorize moves it had already made. By 1978, Belle could search 5,000 positions in a single second [17]. In 1980, Belle implemented several more improvements in the third and final generation. The move generator now included 64 transmitter and receiver circuits which correspond with each space on the board. Transmitters were able to remember the piece on its square and all potential moves it could make, while receivers could detect incoming moves or threats from enemy pieces [17]. The transposition table was now able to store up to 1 megabyte's worth of chess positions. Belle's Alpha Beta algorithm was also now implemented in microcode, which meant that it could control the move generator, position evaluator, and transposition table [17]. With all of these improvements implemented, Belle was now capable of searching upwards of 100,000 positions in a single second. The accumulation of all of these improvements allowed Belle to become the first chess machine to attain a chess master ranking.

3.7 HiTech

Fig. 6. HiTech and its developers Han Berliner and Carl Ebeling [18]

In 1988, the chess machine "HiTech" became the first machine to defeat a grandmaster chess player in a tournament. Much like Belle, HiTech had 64 chips in parallel for the purpose of move generation and position evaluation. The HiTech chess machine also could do quick searching and evaluation by implementing a full width depth first Alpha Beta search algorithm, and this allowed it to be able to search up to 175,000 moves per second [19]. However, what distinguished HiTech from all other chess machines was that it was one of the first machines capable of implementing pattern recognition. Pattern recognition is incredibly useful in chess, as it allows HiTech to ascertain the relationship between all chess pieces on the chess board and use this knowledge to its advantage [20]. Implementing pattern recognition means HiTech was capable of recognizing common, simple checkmate patterns, which increased HiTech's chances of winning chess games in some situations. It also decreased unnecessary search time wasted on the engine's attempt at understanding the properties of the chess board that patterns can easily encode on their own [20]. Another concept which distinguished HiTech from other chess machines was the utilization of "Oracles". An Oracle, in this context, is a repository of knowledge that the chess machine's system uses in order to more efficiently search chess boards and make moves [20]. The Oracle has rules which define goals for both the friendly and opposing chess pieces and the patterns that are needed in order to achieve these goals. Essentially, the Oracle is what enabled HiTech to utilize pattern recognition in chess games.

3.8 Fritz

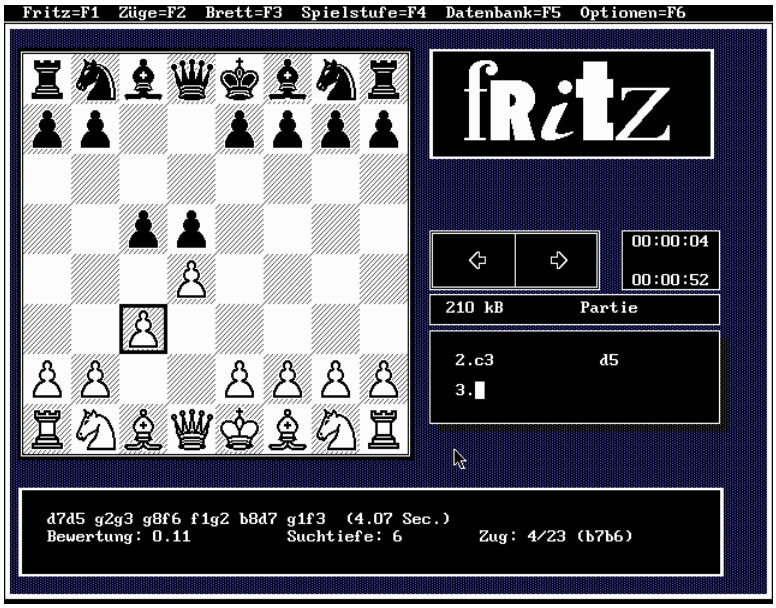

Fig. 7. Fritz chess interface [21]

Fritz was a strong enough chess machine that it was able to defeat one of Deep Blue's earlier versions in a tournament in 1995 with the use of an experimental algorithm called "Null Move Pruning". In chess, a null move is essentially where no chess pieces are moved during your turn. The purpose of null move pruning is to cut on the amount of time spent when a chess machine is searching. Essentially, Fritz will allow the opponent to move twice by simulating a null move, and if Fritz' evaluation function does not return a high value, then that means that there is no threat from an enemy chess piece [22]. Any subtrees which are connected to a null move node where the evaluation function returns a low value, are deemed harmless and irrelevant and are removed from Fritz' search process. This saves on time and computational power that can be directed towards more important areas of the game tree. In Fritz' architecture, move generators, position evaluators, and data structures are all designed and coded as such to utilize the null move search to its maximum potential [22]. Instead of simply adding more parallel processors to increase computational power similarly to Deep Blue, the developers of Fritz used an algorithm which enabled the machine to conduct more efficient searches.

3.9 Deep Blue

Fig. 8. IBM's Deep Blue chess machine [23]

In 1997, Deep Blue became the first chess machine to defeat the world champion of chess, who was Garry Kasparov at the time. This marked the beginning of the concession of the domination of the best human chess players to machinery like the Deep Blue. It consisted of 30 processors, controlling 16 chess chips each [24]. Each chess chip had its own move generator, move stack, evaluation function, and Alpha Beta algorithm hardware. Each chip was capable of searching a minimum of 2 to 2.5 million positions per second, and, with 480 chips, this meant that Deep Blue could search up to 1 billion positions per second [24]. Deep Blue was capable of multiprocessing and utilizing the brute force approach with its powerful hardware unlike any other chess machine in existence. Up until the late 90s, the focus on chess machinery was to increase computational power by adding several parallel processors. Deep Blue and a few other machines were only capable of exhibiting symbolic artificial intelligence in the form of expert systems through the aforementioned "Oracles" which harbored all necessary chess knowledge. After Kasparov was defeated, Deep Blue was dismantled, and the focus on chess machines was no longer on computational power and brute force, but rather, on the utilization of newer and more efficient algorithms and artificial intelligence.

3.10 Stockfish

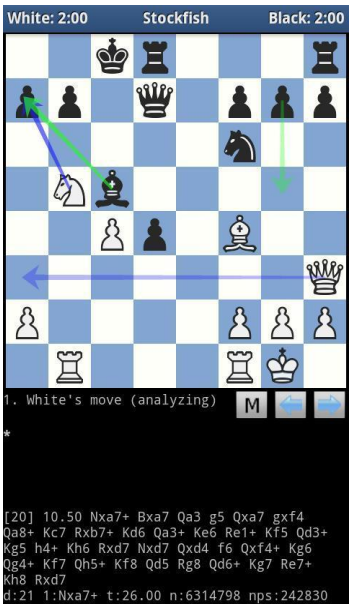

Fig. 9. Stockfish being run on the DroidFish engine [25]

Currently, the strongest chess algorithm in competitive play is Stockfish with an ELO score of 3493 [26]. To better understand Stockfish's strength, the world chess champion Magnus Carlsen has an ELO score 2863, bringing the difference to a massive 630 points [27]. Stockfish's initial release was in 2008, and, since then, the developers have added countless improvements in the form of search algorithms [28]. One of such search algorithms is called the Most Valuable Victim - Least Valuable Aggressor. In this algorithm, a piece capture order in terms of priority is defined according to both the most valuable enemy piece to be captured and the least valuable friendly attacker. The purpose of the algorithm is to not only avoid disadvantageous captures, but to also determine which captures are the most advantageous. In chess, however, captures are often not done without some consequences, since professional players often guard their vulnerable pieces with another piece. There are also scenarios where a professional will purposely give away one of their pieces in exchange for one of their enemy's stronger pieces. Stockfish is capable of determining advantageous and disadvantageous exchanges in these situations with its Static Exchange Evaluation algorithm. The purpose of the Static Exchange Evaluation is to determine the consequences of an exchange of pieces in a single square on the chessboard. The value which represents this exchange is called the "swapoff value". A high swapoff value represents an advantageous exchange while a low value signifies a disadvantageous exchange. In the case of Stockfish, the Static Exchange Evaluation algorithm is optimized to only return a flag which shows whether a position has a high or low swapoff value, so that Stockfish only spends time searching moves which are advantageous to it. These algorithms are only a few from the large pool of algorithms that Stockfish uses. Stockfish also utilizes aforementioned algorithms and methods such as iterative deepening, null move pruning, parallel searching and more.

3.11 AlphaZero

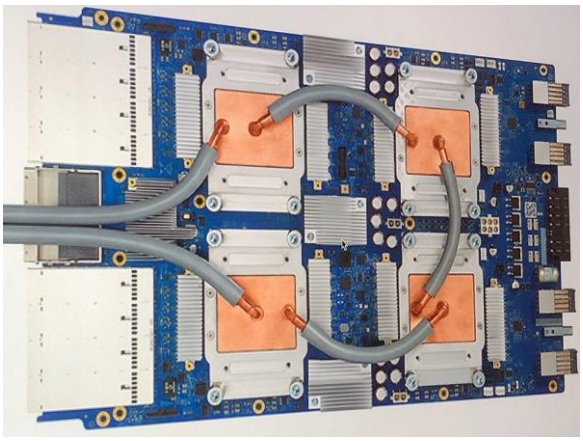

Fig. 10. One of the Tensor Processing Units utilized by AlphaZero [29]

In 2017, chess machines experienced one of the largest revolutions to date with the release of Google's AlphaZero. AlphaZero is the first chess machine to teach itself how to play chess. It utilizes a reinforcement learning algorithm by combining deep learning with the Monte Carlo Tree Search [30]. 5,000 first generation Tensor Processing Units were used to generate self play games [31]. The reinforcement learning algorithm comes into play with these self play games, as it goes through trial and error playing several games with itself, discovering weaknesses within its chess play, and mapping situations and actions in such a way that its able to improve its chess play with every game played. 64 second generation Tensor Processing Units were used in order to train its artificial neural network, which was needed in order for it to better learn how to play the game [32]. Older chess machines which purely implemented brute force approaches such as Deep Blue, used Oracles as a repository of knowledge in which it could base its moves. The Tensor Processing Units, however, are used to improve neural networks which is what AlphaZero bases its moves from. Neural networks serve the purpose of simulating the human brain and the way that neurons communicate with each other inside of it in order to process information like human beings do. The Monte Carlo Tree Search (MCTS) also plays a massive role in AlphaZero's playing capabilities. Essentially, MCTS is a type of search algorithm which is capable of learning based on playouts, which can be defined as the randomized explorations of search spaces [33]. It uses the results of previously successful explorations in order to produce its own game tree, and these game trees are used by chess engines to calculate the optimal move in a chess game. Also, after every playout, the nodes in the MCTS are updated according to the result and how that result was attained, essentially replicating the learning capabilities of human beings and increasing AlphaZero's ability to estimate the values of advantageous chess moves. To prove AlphaZero's strength, Google negotiated a match between AlphaZero and the world's strongest chess engine at the time, Stockfish. AlphaZero won overwhelmingly versus Stockfish with a 28 and 0 winning record [34]. Before this tournament match, AlphaZero had only spent 9 hours training itself how to play chess. Another impressive fact is that AlphaZero was only searching 80,000 positions per second while Stockfish was looking at a massive 70 million positions per second [35]. These numbers convey a single message, and that is that quality is far more important than quantity. AlphaZero covered a smaller search space with far more promising, advantageous moves than Stockfish. With its overwhelming victory, AlphaZero has proven that AI based, self learning chess machinery is capable of dominating, not just Chess, but many other games as well.

4 Discussion

In regards to the criticisms made by the reviewers, there were intentions of performing a quantitative analysis of all chess engines which were aforementioned, but due to a lack of sufficient funding, we were unable to acquire the necessary versions of each chess engine in order to realize the analysis. Originally, we would have had each chess engine play against itself 30 separate times, and record the time (in seconds) and the number of moves needed in order for the engine to have a checkmate. The results of the quantitative analysis would have then been used to determine which chess engine proves to be the most effective in terms of achieving a checkmate.

5 Conclusion

In a matter of 70 years, chess programs were able to dominate a sport which was around for several centuries. The way that chess is played has changed significantly, since the development of chess playing machines and games. Now that these chess AI are available on phones and laptops, there are many children who will be able to learn the game much easier due to easy accessibility of both technology and knowledge. There are also chess machines like AlphaZero that are teaching us new openings in chess theory which have never been seen before. It is fascinating that, originally, it was human beings that were supposed to teach machines how to perform certain tasks. Now, we have machines which are teaching us new ways of playing chess unlike anything that has been seen before. As machines continue to evolve, not just in Chess, but in other areas as well, we should stay open minded and use these paragons as examples so that we can better ourselves too.

References

1. Russell, S. and Norvig, P., 2003. Artificial Intelligence: A Modern Approach. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, p.55
2. n.d. Turochamp Interface. [image] Available at: https://10007969blog.files.wordpress.com/2016/10/turbochamp_glennie_1952b.png
3. Turing, A., 2004. The Essential Turing. Oxford, United Kingdom: Oxford University Press, pp.563-564
4. Levy, D. and Newborn, M., 2009. How Computers Play Chess. Mountain View, California: Ishi Press, p.35.
5. Lih, A., 2005. JOHNNIAC. [image] Available at: https://upload.wikimedia.org/wikipedia/commons/1/1b/Johnniac.jpg
6. Rushton, P. and Marsland, T., 1973. Current Chess Programs: A Summary Of Their Potential And Limitations. INFOR: Information Systems and Operational Research, 11(1), pp.13-20.
7. Walls, B., 2008. Computer Chess History By Bill Wall. [online] Oocities.org. Available at: http://www.oocities.org/siliconvalley/lab/7378/comphis.htm
8. 2018. Mac Hack Bit Display. [image] Available at: https://www.chessprogramming.org/images/5/5f/TV-Ocm.png
9. Wall, B., 2008. Machack Attack. [online] Chess.com. Available at: https://www.chess.com/article/view/machack-attack
10. 2011. Computers And Games: 7th International Conference. Berlin, Germany: Spring Science & Business Media.
11. Wall, B., n.d. Early Chess Computer Programs. [online] Archive.is. Available at: https://archive.is/20120721202324/http://www.chessville.com/BillWall/EarlyComputerChessPrograms.htm
12. Božinovski, S., 2016. Cognitive and Emotive Robotics: Artificial Brain Computing Cognitive Actions and Emotive Evaluations, Since 1981. In: ICT Innovations Conference 2016. Skopje,

p.11.
13. Newborn, M., 2018. Chesstor. [image] Available at: http://archive.computerhistory.org/projects/chess/related_materials/physical-object/3-1%20and%203-3.Chess_4.6_electronic_board_ACM_9_NACCC_Washington_1978_10264526.NEWBORN.jpg
14. Jennings, P., 1978. The Second World Computer Chess Championships. Byte, p.108.
15. Levy, D., 1978. Man Beats Machine!. Chess Life, pp.600-603.
16. Computer History Museum, 2018. Belle Chess Playing Machine. [image] Available at: https://images.computerhistory.org/chess/belle.06230375.jpg?w=600
17. Frey, P., 2012. Chess Skill In Man And Machine. 2nd ed. Berlin, Germany: Spring Science & Business Media.
18. Redick, B., n.d. Hitech Developers. [image] Available at: https://images.computerhistory.org/chess/carnegie-mellon-university.berliner-hans-ebeling-carl.198x.l062302001.cmu.jpg?w=600
19. Newell, A., 1994. Unified Theories Of Cognition. Cambridge, Massachusetts: Harvard University Press.
20. Marsland, A. and Schaeffer, J., 2012. Computers, Chess, And Cognition. Berlin, Germany: Springer Science & Business Media.
21. Marquardt, H., n.d. Fritz Interface. [image] Available at: http://www.septober.de/chess/pics/9104.gif
22. Game-ai-forum.org. 2020. Fritz (ICGA Tournaments). [online] Available at: https://www.game-ai-forum.org/icga-tournaments/program.php?id=27
23. Gardener, J., 2007. Deep Blue. [image] Available at: https://upload.wikimedia.org/wikipedia/commons/b/be/Deep_Blue.jpg
24. Hsu, F., 1999. IBM's Deep Blue Chess Grandmaster Chips. IEEE Micro, 19(2).
25. Österlund, P., Romstad, T., Costalba, M. and Kiiski, J., 2013. Droidfish. [image] Available at: https://upload.wikimedia.org/wikipedia/commons/b/b6/DroidFish.jpg
26. Ccrl.chessdom.com. 2020. CCRL 40/15 - Index. [online] Available at: https://ccrl.chessdom.com/ccrl/4040 [Accessed 20 May 2020].
27. Ratings.fide.com. 2020. Carlsen, Magnus FIDE Chess Profile - Players Arbiters Trainers. [online] Available at: http://ratings.fide.com/card.phtml?event=1503014 [Accessed 20 May 2020]
28. Costalba, M., 2020. Stockfish 1.0 - Talkchess.Com. [online] Talkchess.com. Available at: http://www.talkchess.com/forum3/viewtopic.php?t=24675 [Accessed 20 May 2020].
29. 2019. Tensor Processing Unit. [image] Available at: https://upload.wikimedia.org/wikipedia/commons/b/be/Tensor_Processing_Unit_3.0.jpg
30. 2017. Mastering Chess And Shogi By Self-Play With A General Reinforcement Learning Algorithm. [ebook] Available at: https://www.researchgate.net/publication/321571298_Mastering_Chess_and_Shogi_by_Self-Play_with_a_General_Reinforcement_Learning_Algorithm.
31. Hemsoth, N., 2017. First In-Depth Look At Google'S TPU Architecture. [online] The Next Platform. Available at: https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/.
32. Talkchess.com. 2020. Photo Of Google Cloud TPU Cluster - Talkchess.Com. [online] Available at: http://www.talkchess.com/forum3/viewtopic.php?t=65945 [Accessed 20 May 2020].
33. Chaslot, G., Winands, M. and van den Herik, J., 2008. Parallel Monte-Carlo Tree Search. In: Computers and Games.
34. McGourty, C., 2017. Deepmind'S Alphazero Crushes Chess. [online] chess24.com. Available at: https://chess24.com/en/read/news/deepmind-s-alphazero-crushes-chess [Accessed 20 May 2020].
35. Chess.com. 2017. How Does Alphazero Play Chess?. [online] Available at: https://www.chess.com/article/view/how-does-alphazero-play-chess [Accessed 20 May 2020].
36. Silver, D., Hubert, T., Schrittweiser, J. and Hassabis, D., 2018. *Alphazero: Shedding New Light On The Grand Games Of Chess, Shogi And Go*. [online] Deepmind. Available at: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go [Accessed 22 August 2020].