CHESS BOARD USING JAVASCRIPT AND AI

^ Sahil Mehta (18BCS2232)

"Department of Computer Science and Engineering Chandigarh University Gharuan, Mohali, Punjab, India"

sahilmehta.0sm@gmail.com

ABSTRACT

Chess, which is played between two players on a board, is an intellectual and mental game with its own rules of play that help to enhance and improve the player's mental and intellectual activities, and this game has a large number of players all over the world who are very interested in playing it. This document discusses the computerized information Chess Game. Initially, the game automatically detects for two players to play chess on computer using all of the valid chess rules. Second, to make the game more interesting and entice users to play against the virtual machine, software intellectual force is added.

Key Word: Computer Vision, Artificial Intelligence, AdSense JavaScript, Chess Board, Recommendation, Mind Game.

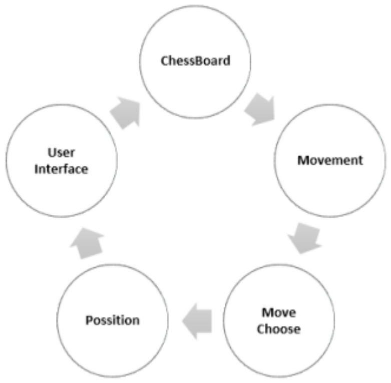
1. INTRODUCTION

Playing chess, a game for two players, is dubbed White and Black. The goal of checkmate is to capture your opponent's king. In thy game, this is called checkmate. Chess is played on a board with 64 squares. Each player begins with 16 pieces, lined up in two rows. The first row is occupied by pieces called pawns. The next row contains a king, a queen, two rooks, two bishops, and two knights. There is no doubt that chess is a game of 'perfect information' since both players are aware of the state of the game world at all times: just looking at the board shows which pieces are alive and where they are situated. Check checkers, Go, Go-Moku, Backgammon, and Othello are also members of the category, but stud poker isn't (you don't know what cards your opponent is holding in his hands). There are several things that need to be done to make the chess game a computer-based, intelligent game. To make a chess game a computer-based, intelligent game, there needs to be some way to represent a chessboard in memory, so that it knows what the state of the game is. • Rules to determine how to generate legal moves, so that it can play without cheating (and verify that its human opponent is not trying to pull a fast one on it!) • A technique to choose the move to make amongst all legal possibilities so that it can choose a move instead of being forced to pick one at random. • A way to compare moves and positions, so that it makes intelligent choices. • Some sort of user interface.

1.1 Preface:

Initial assessment function: Significant research on the analytical solution can be done to improve the playing strength. Although the program's evaluation function has more than ten features, even more can be implemented. [9]There are so many highlights in regular Chess which can be adjusted to inter Chess to increase playing strength. Features representing endgame knowledge, in particular, can be useful because the current evaluation function does not include such a feature. Furthermore, specialists for number of co Chess can indeed be started searching to recover domain expertise that could be used to enhance the evaluation function. Forward pruning: The program does not employ forward pruning techniques. Forward pruning methodologies are used in many regular Chess programs. As a result, it can be investigated whether forward pruning techniques such as Void Transition (Donninger, 1993), ProbCut (Buro, 1995), and MultiCut (Encompasses the following and Marsland, 1999) can improve playing strength..

After chosen the programming language for making the program and the codes manageable and flexible we need to define and create the classes first of all we computerize the chess, second, we are going to create the graphically user interface and finally and the intelligence to the game. The classes are: MainClass.java, King.java, Queen.java, Knight.java, Rook.java, Bishop.java, Pawn.java, interface.java, Algorithm.java. It was an intriguing and very eventually be able for us to see the distinction between the different methodologies by creating a practical stuff instead of just conducting theoretical research..



2. LITERATURE SURVEY

S. Šimoňák *et al.* talk about the motivation behind the project and how imperative algorithms are. The paper also introduces an online platform called Chess AI. They have used Java to prepare the model and deployed it over the internet. What is interesting is that using Chess AI one can see what goes on behind the curtains when say, a backtracking algorithm is run. The author has designed various plug in modules in JavaScript and thus implemented the visualizer. The paper focuses on famous and not to mention the most important sorting algorithms and also renders the conclusion to a survey conducted concerning Chess AI and its effectiveness.

F. Lin *et al.* [2] discussed the future scope of Chess AI and how it can behave learners in the long run. The key feature of the project is that the author has tried to make the user interface as interactive as possible so that there is no gap between the understanding and implementation of an algorithm. The project however is still 'open' and a work in progress. It focuses on the usability and accessibility of the user.

Change, Unlike any other algorithm in Chess AI, 'Easy Chair-Preprint' discusses an interesting approach toward visualization and that is using Pixi.js. The authors discuss how smooth the understanding of data structures and algorithms becomes, once one is well acquainted with the overall effect of algorithms. Java Script is enriched with interesting and complex libraries. The authors have tried to create a visually pleasing GUI where hundreds of numbers are sorted and operated on in front of the user. The paper also takes us into the insightful journey of why Pixi.js was implemented, it states that the development of GUI and interactivity translates to the excellent user as well as builder experience. The paper further discusses how quick sort has been implemented and what the structure of the code looks like.

3. METHODOLOGY

Let's explore some basic concepts that will help us create a simple chess AI:

1. Move-generation
2. Board evaluation
3. Minimax
4. Alpha beta pruning.

At each step, we'll improve our algorithm with one of these time-tested chess-programming techniques. I'll demonstrate how each affects the algorithm's playing style.

3.1 Move generation and board visualization

We'll use the chess.js library for move generation, and chessboard.js for visualizing the board. The move generation library basically implements all the rules of chess. Based on this, we can calculate all legal moves for a given board state.

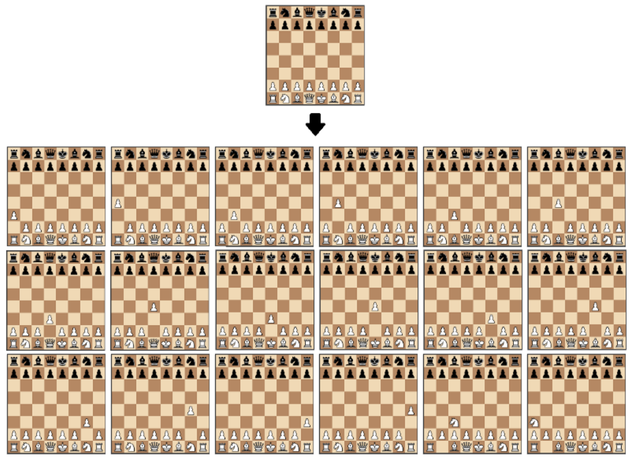


Fig 3.1.1

3.2 Position evaluation

Now let's try to understand which side is stronger in a certain position. The simplest way to achieve this is to count the relative strength of the pieces on the board. With the evaluation function, we're able to create an algorithm that chooses the move that gives the highest evaluation:

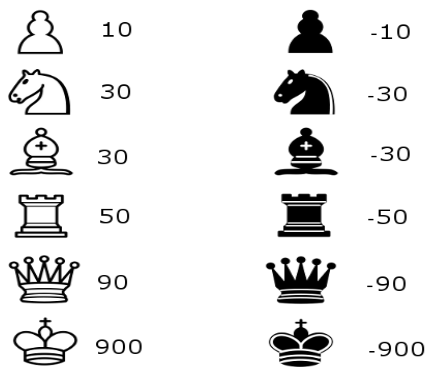

10
-10
30
-30
30
-30
50
-50
90
-90
900
-900

Fig3.2.1

The only tangible improvement is that our algorithm will now capture a piece if it can.

3.3 Search tree using Minimax

Next we're going to create a search tree from which the algorithm can choose the best move. This is done by using the minimax algorithm. In this algorithm, the recursive tree of all possible moves is explored to a given depth, and the position is evaluated at the ending "leaves" of the tree.
After that, we return either the smallest or the largest value of the child to the parent node, depending on whether it's a white or black to move. (That is, we try to either minimize or maximize the outcome at each level.)

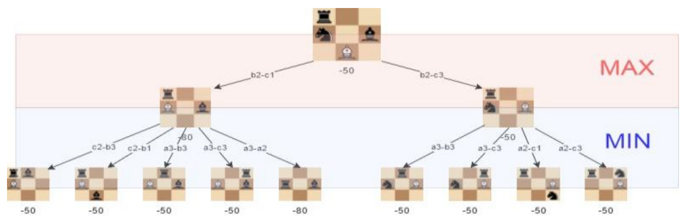


Fig 3.3.1

3.4 Alpha-beta pruning

Alpha beta pruning is an optimization method to the minimax algorithm that allows us to disregard some branches in the search tree. This helps us evaluate the minimax search tree much deeper, while using the same resources.
The alpha-beta pruning is based on the situation where we can stop evaluating a part of the search tree if we find a move that leads to a worse situation than a previously discovered move. The alpha-beta pruning does not influence the outcome of the minimax algorithm — it only makes it faster. The alpha-beta algorithm also is more efficient if we happen to visit first those paths that lead to good moves.

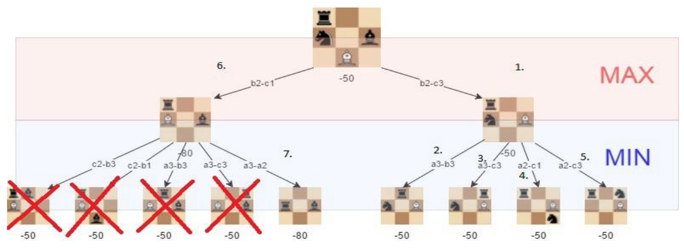
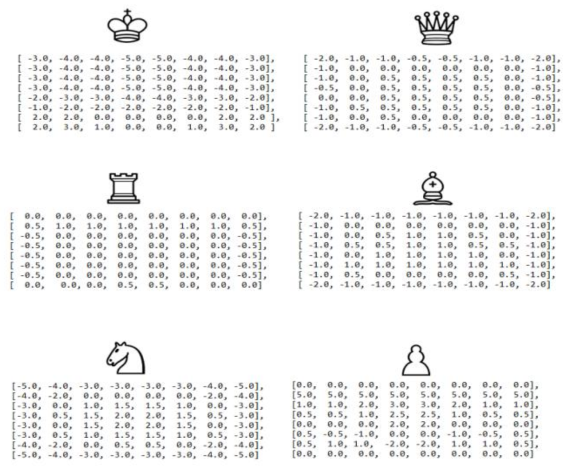


Fig3.4.1

3.5 Improved evaluation function

The initial evaluation function is quite naive as we only count the material that is found on the board. To improve this, we add to the evaluation a factor that takes in account the position of the pieces. For example, a knight on the centre of the board is better (because it has more options and is thus more active) than a knight on the edge of the board.



```
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
[ -1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
[  2.0,  2.0,  0.0,  0.0,  0.0,  0.0,  2.0,  2.0 ],
[  2.0,  3.0,  1.0,  0.0,  0.0,  1.0,  3.0,  2.0 ]
```

```
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -0.5,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[  0.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[ -1.0,  0.5,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]
```

```
[  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[  0.5,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[  0.0,   0.0, 0.0,  0.5,  0.5,  0.0,  0.0,  0.0]
```

```
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  1.0,  1.0,  0.5,  0.0, -1.0],
[ -1.0,  0.5,  0.5,  1.0,  1.0,  0.5,  0.5, -1.0],
[ -1.0,  0.0,  1.0,  1.0,  1.0,  1.0,  0.0, -1.0],
[ -1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0],
[ -1.0,  0.5,  0.0,  0.0,  0.0,  0.0,  0.5, -1.0],
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]
```

```
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
[-4.0, -2.0,  0.0,  0.0,  0.0,  0.0, -2.0, -4.0],
[-3.0,  0.0,  1.0,  1.5,  1.5,  1.0,  0.0, -3.0],
[-3.0,  0.5,  1.5,  2.0,  2.0,  1.5,  0.5, -3.0],
[-3.0,  0.0,  1.5,  2.0,  2.0,  1.5,  0.0, -3.0],
[-3.0,  0.5,  1.0,  1.5,  1.5,  1.0,  0.5, -3.0],
[-4.0, -2.0,  0.0,  0.5,  0.5,  0.0, -2.0, -4.0],
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]
```

```
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0],
[1.0,  1.0,  2.0,  3.0,  3.0,  2.0,  1.0,  1.0],
[0.5,  0.5,  1.0,  2.5,  2.5,  1.0,  0.5,  0.5],
[0.0,  0.0,  0.0,  2.0,  2.0,  0.0,  0.0,  0.0],
[0.5, -0.5, -1.0,  0.0,  0.0, -1.0, -0.5,  0.5],
[0.5,  1.0, 1.0,  -2.0, -2.0,  1.0,  1.0,  0.5],
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0]
```

Fig3.5.1

4. IMPLEMENTATION METHOD

This section explores fully chess implementation details, we tried to explain the whole implementation classes and their methods.

As we mentioned in the previous section intelligent chess in graphically user interface needed a computer programming language for implementation, therefore we have selected java which is fully object-oriented and runs in any plate form that will lead the chess to run in any plate form.
After choosing the programming language for making the program and the codes manageable and flexible we need to define and create the classes, first of all, we computerize the chess, second, we are going to create the graphical user interface, and finally, the intelligence to the game. The classes are: MainClass.java, King.java, Queen.java, Knight.java, Rook.java, Bishop.java, Pawn.java, interface.java, Algorithm.java.
We gave some information in the last section about these classes, this section will focus on more details, especially the function and methods of the classes.

MainClass.java Class

This is the first class of the project which consists in:

- main(String[] args) Is the first function of the main class, that makes the bases of the program, in other words, this is the starting function of the project.
- posibleMovesW() This function is for returning the valid moves of white pieces.
- PosibleMovesB() This is the same above functions and generating the valid moves for Black pieces.
- makeover(String move) This is function is for doing a move for Black Pieces.
- makeover(String move) this is the same as the last function it does a move for White Piece.
- undoMoveB(String move) this function does the undo action for Black pieces.
- undoMoveW(String move) this function does the same thing as the last function for white pieces.

King.java Class

This class includes the whole valid rules for the piece of a king in the game.

- possible(int i) this function return all valid moves for white pieces.
- posibleKB(int i) this function return all valid moves for black pieces.
- kingSafeW() this function is for checking the state of the white king.
- kingSafeB() this function is for checking the state of the black king.

Queen.java Class

This class includes the entire functions depending on the Queen in the project.

• posibleQW(int i) this function return all the valid moves for the queen.
• posibleQB(int i) this function return all the valid moves for the queen.
Knight.java Class
This class includes the whole rules concerning the knight.
• posibleNW(int i) this function returns all possible and valid moves of the white knight.
• possibleNW(int i) this function returns all possible and valid moves of the black knight.
Bishop.java Class
This class consists of the all rules of the Bishop.
• posibleBW(int i) this function return all valid moves of the white Bishop.
• posibleBB(int i) this function returns all valid moves of the black Bishop. 6.7 Rook.java Class This class explores the whole rules related to Rook.
• posibleRW(int i) the white rooks all valid move returns by this function.
• posibleRB(int i) this is the same function for the black. 6.8 Pawn.java Class This class includes the all rules related to the pawn.
• posiblePW(int i) this function returns the whole valid moves for the white pawn.
• posiblePB(int i) this function returns the whole valid moves for the black pawn.
InterFace.java class
This class consists of the whole materials concerning the graphical environment of the project.
• paintComponent(Graphics g) this function generates the chessboard and its elements for graphically user interface.
Algorithm.java Class
This class includes all functions related to the intelligence of the chess game.
• alphaBeta(int depth, int beta, int alpha, String move, in player) this function implement the alpha beta pruning algorithm.
• rateAttack() this function is for evaluation and identification of risks.
• rate material() this function for identification of the remaining pieces on the board.
• rateMoveablity(int strength, int depth, int material) this function is for identifying all the moves related to the computer game.
• rare positional(int material) this is for finding the location of computer movement pieces.
• rating(int list, int depth) This function rates the whole piece in each possible move of the computer.
Here we can see how the dashboard work and what are the other tools in the project.

As you can see from the above screenshot, we are done with the main functionality of our chessboard with AI & JavaScript i.e our main page (Our Front End). And now we are working on the other functionalities that we have added to this main page, to calculate and provide the best analysis of Dynamic values using javascript & AI. The coding for representing the Dynamic Data, through our other functionalities is under process. Moreover, we look forward to making complex things simpler.

5. RESULT AND DISCUSSION

1. Agency: Incorporate information, objectives, and constrictions that the decision-maker typically considers.[19] when making the choice
2. Viewpoint: Analysis from a situation of arena and intellectual capital, as well as comprehension.
provide context for the process that produced it
3. Reliability: Consider the complexities of the surroundings in which it will be used.
4. Objectivity: Avoid predictors who are biased.
5. Transparency: Be straightforward, peer-reviewed, or tested to ensure that undesirable biases are not introduced. Inadvertently infiltrated
6. Sincerity: Effectively manifest attempts to measure of composure and "why" messages (ideally in written form intuitive language) elucidating why a particular algorithmic indicator as it originally is.[18]

This is the interface of the project.
Algothimic expansive improvement
Here we can see that how the dashboard work and what are the other tools in the project. As you can understand, we are done with the main functionality of our chess board with AI & JavaScript i.e., our main page (Our Front End). And now we are working on the other functionalities that we have added in this main page, to calculate and provide best analysis of Dynamic values using JavaScript & AI[17]. The coding for representing the Dynamic Data, through our other functionalities is under process. And moreover, we are looking forward to make complex things simpler.

5.1 Data flow chart of the proposed work

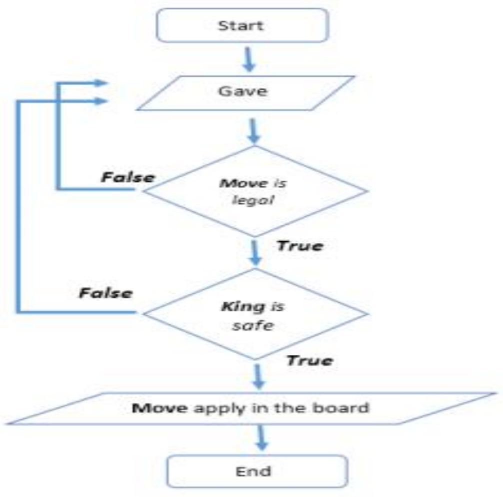


Fig 5.1

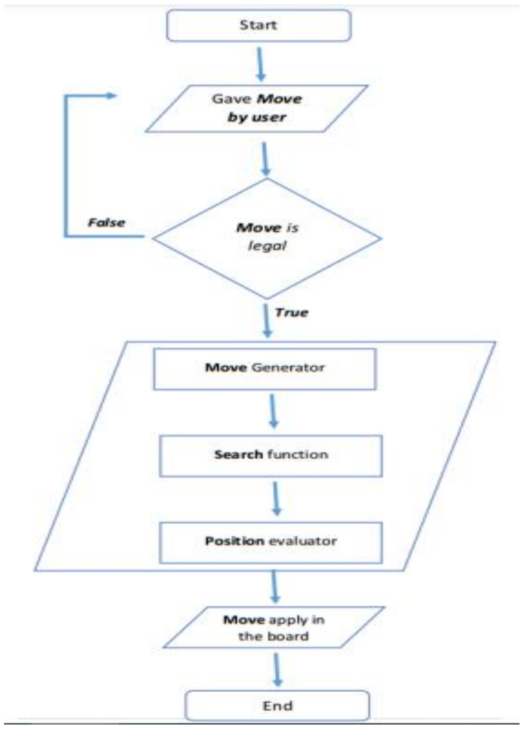


Fig 5.2

6. CONCLUSION AND FUTURE SCOPE

The main focus of this chapter is on the work has done and future enhancement of the project, the first section discusses the conclusion and summary of the whole Final Project, and the second section is dedicated to the future enhancement of the project. The goal of this Final Project was to create a practical and usable product that could be considered software and even an educational tool. Modeling a real-life scenario, which is a chess game in the current case, without any restrictions and based on the modeler's understanding was another objective of this Final Project was to analyze two agent-based approaches. It was a fascinating and quite helpful experience for us to observe the differences between the two approaches by producing practical work rather than conducting just theoretical research. The users will observe and realize the fact that it is a very open-ended model, allowing users to interact with it using their imagination, which was the main idea behind all this effort. Hopefully, however, it has managed to lay the groundwork for further study of narrative in video games.

Finally, the program must have some way of assessing whether a given position means that it is ahead or that it has lost the game.

This evaluation depends 23 heavily upon the rules of the game: while ”material balance” (i.e., the number and value of the pieces on the board) is the dominant factor in chess because being ahead by as little as a single pawn can often guarantee a victory for a strong player, it is of no significance in Go-Moku and downright misleading in Othello, where you are often better off with fewer pieces on the board until the very last moment.

Actually, The strength of even a simple chess-playing algorithm is that it doesn’t make stupid mistakes. This said, it still lacks strategic understanding with the methods I introduced here , we’ve been able to program a chess-playing-algorithm that can play basic chess. The “AI-part” (move-generation excluded) of the final algorithm is just 200 lines of code, meaning the basic concepts are quite simple to implement. Some further improvements we could make to the algorithm would be for instance:

1. Move-ordering

2. Faster move generation

3. End-game specific evaluation.

For future references, We should introduce two things to computer for making the game intelligent, which will make the game to do an optimal move (a move which grant the most gain and give the most harm to opponent).
• A technique to choose the move to make amongst all legal possibilities, so that it can choose a move instead of being forced to pick one at random.
• A way to compare moves and positions, so that it makes intelligent choices
Chess game computerization needed two things a board representation an pieces movements generation as we discussed in previous section. For making the intelligent we need two other things as well, first legal moves creation and a random move selection, second move selection after evaluation of all generated moves intelligently.

7. REFERENCES

[1] S. Šimoňák, "Using algorithm visualizations in computer science education." Central European Journal of Computer Science, vol. 4, no. 3, pp. 183-190, 2014.

[2] F. Lin, A. Dewan and V. Voytenko, "Open Interactive Algorithm Visualization," 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), pp. 1-4, 2019.

[3] Risi, Sebastian, and Mike Preuss. "From chess and atari to starcraft and beyond: How game ai is driving the world of ai." *KI-Künstliche Intelligenz* 34.1 (2020): 7-17.

[4] Maharaj, Shiva, Nick Polson, and Alex Turk. "Chess AI: Competing Paradigms for Machine Intelligence." *Entropy* 24, no.4 (2022): 550.

[5] Parteek & Kottawar, Vinayak & Deshmukh, Pramod. (2021). AlgoAssist: Algorithm Visualizer and Coding Platform for Remote Classroom Learning. 1-6.

[6] Untold History of AI: When Charles Babbage Played Chess With the Original Mechanical Turk, spectrum.ieee.org

[7] Burmeister, Jay, and Janet Wiles. "The challenge of Go as a domain for AI research: a comparison between Go and chess." In *Proceedings of Third Australian and New Zealand Conference on Intelligent Information Systems. ANZIIS-95*, pp. 181-186. IEEE, 1995.

[8] Bringsjord, Selmer. "Chess Is Too Easy." *Technology Review* 101, no. 2 (1998): 23-28.

[9] Hassabis, Demis. "Artificial intelligence: chess match of the century." *Nature* 544, no. 7651 (2017): 413-414.

[10] Marsland, T. Anthony, and Jonathan Schaeffer, eds. *Computers, chess, and cognition*. New York: Springer, 1990.

[11] McIlroy-Young, Reid, et al. "Learning personalized models of human behavior in chess." *arXiv preprint arXiv:2008.10086* (2020).

[12] Levinson, Robert, et al. "The role of chess in artificial intelligence research." *ICGA Journal* 14.3 (1991): 153-161.

[13] Rasskin-Gutman, Diego. *Chess metaphors: Artificial intelligence and the human mind*. MIT Press, 2009.

[14] Yen, Shi-Jim, et al. "Design and implementation of Chinese dark chess programs." *IEEE Transactions on Computational Intelligence and AI in Games* 7.1 (2014): 66-74.

[15] Sanjaya, R., Wang, J., & Yang, Y. (2021). Measuring the Non-Transitivity in Chess. *arXiv preprint arXiv:2110.11737*.

[16] Matsubara, Hitoshi, Hiroyuki Iida, and Reijer Grimbergen. "Chess, Shogi, Go, natural developments in game research." *ICCA journal* 19.2 (1996): 103-112.

[17] Mikhaylova, I. V., Makhov, A. S., & Alifirov, A. I. (2015). Chess as multicomponent type of adaptive physical culture. *Theory and practice of physical culture*, (12), 56.

[18] Hu, Y., & Lv, R. (2021, May). Chess AI with Different Behavioral Tendencies and Its Application. In *2021 2nd International Conference on Artificial Intelligence and Information Systems* (pp. 1-5).

[19] Fransson, Henric. "AgentChess: An Agent Chess Approach." (2003).

[20] McCarthy, John. "AI as sport." (1997): 1518-1519.