Minimax Algorithm for a King and Rook vs King Chess Endgame

Adrijan Božinovski and Filemon Jankuloski
School of Computer Science and Information Technology
University American College Skopje
Skopje, Macedonia

bozinovski@uacs.edu.mk
filjankuloski@gmail.com

Abstract—**In this paper, we will take a look at an Artificial Intelligence based chess program, which portrays a White King and Rook vs Black King endgame scenario. First, we will take a look at the evolution of this chess program, and how it progressed from being purely algorithmic to employing Artificial Intelligence. Next, we will briefly explore how the main program works and scrutinize all of its possible functions. Moreover, the version of the program which obtains all relevant statistics as well as the circumstances surrounding their attainment will also be discussed. Finally, we will conclude the paper by analyzing the results of the exhaustive testing and showcasing a hypothesis derived from the aforementioned results.**

Keywords—Chess, Artificial Intelligence, Statistics, King and Rook vs King endgame

I. Introduction

A. Goal of the Project

Artificial Intelligence (AI) has become increasingly prevalent in our current society. We can find AI being used in many aspects of our daily lives, such as through the use of smartphones and autonomous driving vehicles such as the Tesla [1]. Leading textbooks on AI define it as the study of "intelligent agents", which can be represented by any device that perceives its environment and takes actions that maximize its chances of achieving its goals [2]. In the context of this paper, the intelligent agent is a chess-playing AI, which implements the Minimax algorithm and controls the moves of both Black and White in a particular endgame scenario. The aforementioned endgame scenario that we will be exploring is a White Rook and White King, versus a Black King, both of which are controlled by AI. Such a program has already been created, and it was the first Macedonian chess program, which was created by Stevo Božinovski in 1969, and was written in Fortran for the IBM 1130 computer [3]. However, in the case of this paper, the AI is written in Java and uses Netbeans as its IDE. The goal of this project is to showcase the Minimax algorithm's capabilities and to test if increasing the number of steps - i.e., the depth of the Minimax/Maximin tree - will improve the AI's level of chess play. The evolution of this chess AI, the program, the relevant methods, the results, as well as how they were produced, will be showcased in a subsequent section.

B. Evolution of the Current Chess Program

This chess program's first version was created by Stevo Božinovski in 1969 [4]. This program used several different methods for specific purposes. "DATSW" was used to plot the chess board, "POTEZ" was used to determine the legality of the human move, "POZIC" was the algorithm which served as both a position analyzer and a move generator, and "MATIR" determined if the state of the chessboard was in checkmate. All versions following the first version were made by Filemon Jankuloski and Adrijan Božinovski.

The second version of this program was finished in 2021. This program was purely algorithmic and needed immense amounts of conditionals to cover all possible situations, but was far more specialized than the first version. It also had a position evaluator just for White's moves, since Black was programmed to be played by a human player. The second program also was capable of instantaneously resetting the chessboard to the initial position at any point during a game, generating new and valid random starting positions, creating an enumeration for the state of the chessboard and its pieces, and calculating the value of the state of the board, based on the positions of all pieces on the board.

The third version of this program was finished in 2022, and was successfully converted to a heuristics-based program. This version had position evaluators for the moves of the White King and White Rook separately, while Black was played by a human player. Inside the position evaluators were different sets of criteria, and each individual factor in these criteria could increase or decrease the total value. The lower the total value, the more preferable the White Player (i.e., the AI) would find the corresponding move for that value. The heuristic used in this program resembled that of an A* Search algorithm. The standard A* Search displays a metric of the current node in the search from the starting point, as well as an estimate of the cost for a future node that would be searched [5]. However, in the case of this program, the initial point could be described as the state of the board before a move was made, and nodes could be described as every potential square that the White pieces could move to. The A* Search algorithm uses the equation $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of traversing from one node to another, $h(n)$ is the heuristic approximation of the node's value, and $f(n)$ is the final cost [6]. In this version of the program, however, $g(n)$ was the value of the current position, $h(n)$ was calculated through position evaluators, and $f(n)$ was the final cost of the move. The goal of the program was for White to deliver a checkmate, by moving to squares where the total cost would be, by design, the lowest.

The final and most current version of the program was finished in March 2023, and employs Artificial Intelligence through the utilization of the Minimax algorithm. The Minimax algorithm is an algorithm where White's moves are calculated based on Black's potential future moves, whereas Black's moves are calculated based on White's potential future moves; since Black's goal and its means of calculation are opposite to that of White, it is said that Black utilizes the Maximin algorithm.

In regards to the Minimax algorithm, a "step" represents how many moves the algorithm is looking ahead, whereas a "step" in the Minimax/Maximin tree pertains to a full move in chess, i.e., when a move has been made by both White and Black. The more steps are implemented into the algorithm, the better the AI's moves are expected to be. A position evaluator is also used in this version of the code; however, it is a single evaluator where the same criteria are shared by every piece, meaning the Black King, the White King, and the White Rook. A brief overview of the Minimax algorithm's implementation, as well as the position evaluator, will be given in the subsequent section.

II. THE MAIN PROGRAM

A. *The User Interface of the Program*

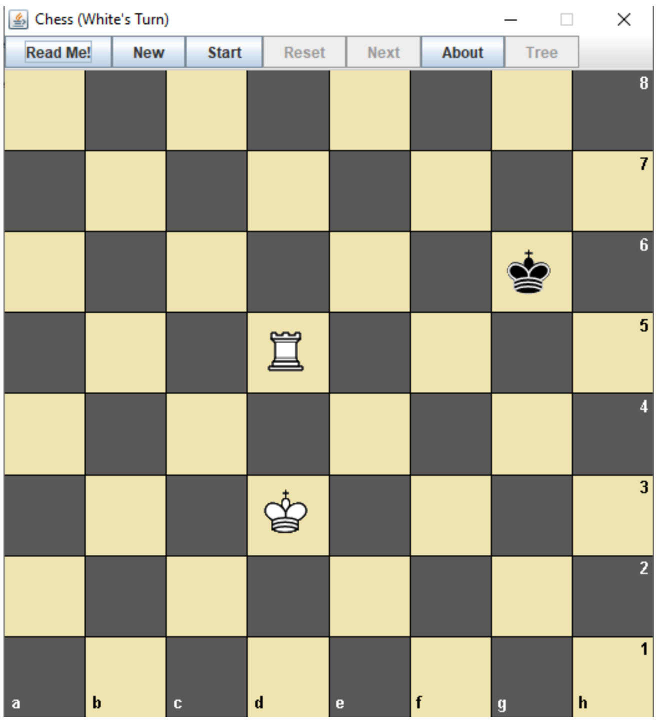

Fig. 1 User Interface when the Program is Executed

The user interface for the chess program looks much like a normal chess board, as shown in Figure 1, with files being labeled "a" to "h" and ranks being labeled "8" to "1". The frame of the window shows whether it is White's or Black's turn, as well as the enumeration of the current state of the board. There are also buttons with different functionalities. Clicking the "Read Me!" button will open a window which explains to the user how the program and all of the buttons function. Clicking the "New" button will generate a new random legal starting position. Clicking the "Start" button will commence the game with a move from the White pieces. Clicking the "Reset" button will reset the chess pieces back to their initial position before the "Start" button has been clicked. Clicking the "Next" button will initiate either Black's or White's next move, depending on which side's turn it is. Clicking the "About" button will open a window where users can read a short description about the creators of this program.

Clicking the "Tree" button will open a new window with a tree diagram, where the depth is equal to double the amount of steps, plus the root node. The root node represents the initial position of the board before a move is made. If it is white to move, a step consists of all potential moves which can be made by White, followed by all the potential moves which can be made by Black starting from those potential White moves - this is what is known as the Minimax tree. A Maximin tree is analogous to the Minimax tree, except that it is generated upon, i.e., after, Black's move. Each node on the tree diagram is represented by a string of numbers and characters (e.g., 120k12R35K65), which, in their respective order, represent: the value of the position, followed by the coordinates of the Black King, White Rook, and White King.

B. *The Minimax Algorithm*

The Minimax algorithm begins with the root node (i.e., the 0th level), which is represented by the enumeration of the initial state of the chess board. Next, from the root node, the nodes in the 1st level consist of all legal White moves. The enumerations of these moves, as well as their values, are stored inside a linked list. From every White move, every legal Black move is stored inside a linked list on the 2nd level, including the Black move's: enumeration, value, and a value known as the "origin". The origin represents one of the potential White moves that can be made which follow the root node. At this point in the algorithm, one step has been completed. In the third and fourth level, all legal White moves and Black moves, respectively, are recorded inside a linked list, which includes: the enumeration, the value, and the same origin node from the 1st level. The process of sorting out the nodes begins once there is a linked list which contains every node of the last level on the tree. Nodes on every level of the tree have a root node, with the exception of the 0th level. Nodes which share the same root node, including the root node itself, is what comprises what is known as a "subtree".

As an example let us say that we are working with a Minimax tree with only 1 step. The 2nd level (last level of the tree) consists of legal Black moves, and for each subtree we want the node with the highest value, otherwise known as the maximizing step. Afterwards, we will switch to the minimizing step, and what we will want will be the minimum value from each subtree instead. In the end, there will be only one node left which has passed both the maximizing and minimizing step, and the origin of this node will refer to the enumeration of the move that will be made. It should be noted that, when there is only one subtree left in the Minimax algorithm, and there are more than 2 nodes which share the lowest value, one of those multiple nodes is chosen at random.

The analogous approach is used when it's Black to move, except that the initial root of the tree is the position before Black's move, whereas the subsequent nodes represent all possible positions arising from Black's moves, and their subnodes are all possible positions arising from White's moves. In this case, Black wants to select a move which will result in the greatest minimum value that White could choose, so therefore Black employs the Maximin algorithm.

C. *Position Evaluator*

Position evaluators are used to assign values to nodes. All pieces share a single evaluator so the same criteria will be applied regardless of whether the piece is White or Black. There are only 2 values which are taken into consideration and they are known as “freeSquares” and “endgameValues”, whereas the total value of a position is obtained as the sum of these two values.

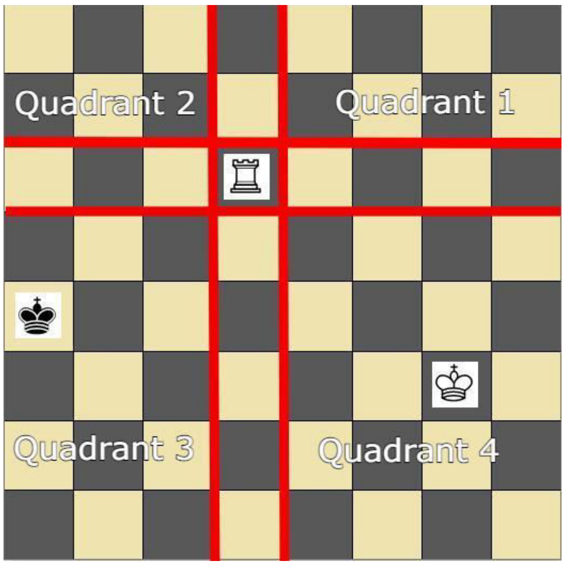

Fig. 2 Quadrant Positioning Inside the Chess Board

“freeSquares” is the amount of squares that the Black King is able to move to. Similarly to Figure 2, the chess board is divided into 4 separate quadrants, and each quadrant is relative to the White Rook’s position on the board, which is represented by the intersection of the x and y axes. Quadrant 1 is to the upper right of the rook, quadrant 2 is to the upper left, quadrant 3 is to the lower left, and quadrant 4 is to the lower right. Depending on which quadrant the Black King resides in, there are corresponding formulas which the program uses to find how many squares the Black King is restricted to, depending on whether the Rook restricts it from the top, bottom, left, or right. These criteria can also be calculated differently depending on several conditions, including: when the board is in a state of check with opposition, when the board is in a state of check without opposition, when the White Rook is not blocked off by the White King, when the White Rook is blocked off by the White King with no passage beyond the Rook’s line of attack, and when the White Rook is blocked off by the White King with a passage beyond the Rook’s line of attack.

“freeSquares” is initially equal to 0. A position invokes the method which calculates free squares only when the state of the board is not checkmate or stalemate, because, by definition, the amount of free squares that the Black King can move to will be 0 in both situations. It is also important to note that the White King can subtract anywhere between 0-8 squares from the total free squares count, but only when it is within the same quadrant as the Black King. It has been shown that the “freeSquares” value can be equal to anywhere between 0 and 49.

“endgameValues” are values which are assigned only when flags (i.e., Boolean variables) light up (i.e., obtain “true” values) for one of the five endgame situations accounted for in this program, namely: checkmate, stalemate, 50 move rule draw, threefold repetition draw, and rook capture draw. Checkmate (considered a win for White) occurs when the Black King is put in a situation where its capture is unavoidable, i.e., when it is in a state of check and has no free squares to move to. Stalemate (considered a draw) occurs when it is Black’s turn to move, but it can only stay at the square where it currently resides, otherwise it would move into a state of check, i.e., when it has no free squares to move to but is not in a state of check. A 50 move rule draw occurs when both the Black and White pieces have made 50 moves and no irreversible move has been made (such as capture of the White Rook, checkmate or stalemate). A threefold repetition draw occurs when the game arrives at the same position of the board 3 total times. A rook capture draw occurs when the White Rook is captured by the Black King, and consequently, there are only 2 Kings left on the board. In the case of the flags lighting up for stalemate, 50 move rule draw, threefold repetition draw, and rook capture draw, “endgameValues” would be equal to 1000, whereas for checkmate “endgameValues” would be equal to -1000.

Since White wants to restrict the Black King to progressively smaller amounts of free squares, so as to eventually deliver checkmate (and thus arrive at the -1000 “endgameValues” value position), White’s goal is to choose moves with the smallest possible values for both “freeSquares” and “endgameValues” whenever possible, which is why White employs the Minimax approach. Conversely, Black wants to maximize the number of free squares available and eventually achieve any of the 4 draw scenarios (and thus arrive at any 1000 “endgameValues” value position) which is why Black employs the Maximin approach.

III. The Process for Producing Results

A. *Attaining All Legal Initial Positions*

Before testing the difference between 1-step and 2-step Minimax, first, all legal chess positions must be attained. The program is capable of displaying an enumeration for each position, and the enumeration is calculated such that each piece is given a numerical value based on which field on the chessboard it is placed (from 0 to 63, inclusive). Then, each piece’s value is multiplied either by 1, 64, 64^2 (in this paper, the pieces which get those factors are the Black King, the White Rook and the White King respectively) and the sum of all those values gives the enumeration of the position reached. Given the formula above, one can calculate that there are a total of 262,144 total enumerations representing every single possible combination of coordinates for the White King, White Rook, and Black King. However, not all of these positions are legal according to the rules in chess, since, for example, placing two pieces on the same square is illegal in chess.

A program was created in Netbeans IDE known as “JavaApplication13” which was assigned the task of creating a file named “legalpositions.txt”. After creating the file, a for loop is initiated, starting from 0 up until 262,143. These values are converted from enumerations into coordinates, and then are put through a filter to check for legality. The conditions for legality are that none of the pieces are to have the same coordinates and that the game cannot start in a position of check or checkmate (an initial stalemate position is allowed, since it is always White to move first, so White could make a move to alleviate the stalemate). Once the position has been determined to be legal, its enumeration is written into the "legalpositions.txt" file. All legal positions in the file are delimited by a newline, and the total number of all legal positions is equal to 175,168.

B. Testing All Legal Positions

A different program from "JavaApplication13" was assigned the task of testing all legal positions, and it was named "JavaApplication12". This program is similar to the main chess program, with some substantial differences. Firstly, all GUI aspects of the program have been removed in this program, in order to increase performance. Secondly, the program creates both a directory named "LegalPositionTests" and 175,168 files which are placed inside the aforementioned directory. These files are all named after a legal enumeration (e.g., "194.txt") and are created before the legal enumeration is put through a simulation game (i.e., a chess game which runs on its own, without the need for human input). Moreover, there are new functions added to this program for the sake of writing test results into the files and these results will be explained in detail subsequently.

The first thing to be written into the file is the starting position of the legal enumerations, represented in Forsyth-Edwards notation, i.e., FEN. FEN is a way of representing the chessboard and the position of all the pieces. For example, the Forsyth-Edward notation of Figure 1 would be "8/8/6k1/3R4/8/3K4/8/8". The Black King, White Rook, and White King are written as "k", "R", and "K" respectively. Numbers represent the amount of squares without pieces on them from left to right. The first 2 rows are empty, so they are simply written as "8". However, because there is a Black King on the 3rd row, this row is written out as "6k1" because there are 6 free squares behind the Black King and 1 in front of it.

The second thing to be written into the file is the standard chess notation of the moves taken during the entire game. For example, if we were to write the first move in chess notation, it could look something like "1. Kc2 Ka7". The number represents which move one would be reading and "Kc2" and "Ka7" represent White's move and then Black's move respectively. If we were to break down the meaning of "Ka7", "K" would be the piece moved and "a7" would be where it was moved to. So in the case of the first move, the White King moved to the file "c" and rank "2" while the Black King moved to the file "a" and rank "7". Regardless of whether the White King or the Black King is moved, all chess piece abbreviations are written with their respective capital letters, so King is always written as "K". There are also special cases when it comes to chess notation. When White delivers a check, a "+" is attached to the end of its move. When White delivers a checkmate, a "#" is written at the end of its move. When White delivers a stalemate, a "$" is written at the end of its move. If White or Black's move causes a threefold repetition draw, or Black's move causes a 50-move-rule draw or a rook capture draw, then "½-½" is attached to the end of the respective move. Additionally, in the case of a rook capture draw, an "x" would be placed between the abbreviation of the piece and where the piece moved to perform the capture (e.g., "Kxc7").

The third thing to be written is the amount of moves it takes for the game to come to a conclusion. The fourth thing to be written is the outcome of the game, and this would be: "checkmate", "stalemate", "50-move-rule draw", "threefold repetition draw", or "rook capture draw", as applicable.

Since each file contains the results of both the 1-step and 2-step simulation game from the given starting position, "1-Step Minimax/Maximin" and "2-Step Minimax/Maximin" is written before the aforementioned results are printed out, for the sake of differentiation.

C. The Circumstances Surrounding the Testing and Gathering of the Results

Although there were 175,168 different positions to be tested, because they were tested in 1-step and 2-step, this means the actual number was a staggering total of 350,336 simulation games. This immense number of games required a powerful computer if the tests were to be complete within a reasonable amount of time. Luckily, all tests were completed within approximately 40 hours. The computer used to complete these tests came equipped with an Intel Core i7-4790K processor, 32 gigabytes of RAM memory, x64-based processor, and had Windows 10 Pro installed on it. The tests were not done in a single go, and instead were separated into two halves of 87,584 positions each. Both of them were carried out during 2 separate days, each taking roughly 1,200 minutes to run to completion. After the "LegalPositionTests" directory was filled with all 175,168 files, it was placed within another project folder, to be used by another application named "JavaApplication11". This application was used to read every file within the directory and compiled results for the 1-step and 2-step algorithms separately, such as: the percentage of each outcome obtained, the average number of moves per game and the percentage increase and/or decrease in these statistics from 1-step to 2-step.

IV. Results and The 7-Step Hypothesis

The results for 1-step and 2-step are completely separate and the assumption before the testing was that 2-step would incur better results than 1-step, as viewed from White's perspective. Since all 175,168 positions were tested exhaustively, the results should be mostly accurate. However, since there is a possibility that the randomization of moves with tied values could affect results to some degree, there is some doubt that the results are fully accurate. A comprehensive set of test results would require a minimum of 30 tests of each initial starting position, for both 1-step and 2-step, which was not done because of the limitation of the capabilities of the available hardware, as well as time constraints.

From the obtained results, in 1-step, 48.27% of games resulted in checkmates and the other 51.73% resulted in 50-move-rule draws. The average number of moves per game was 37 moves. In 2-step, 84.25% of games resulted in checkmates and the other 15.75% resulted in 50-move-rule draws. The average number of moves per game was 23 moves. The number of checkmates increased by 74.54%, the number of 50-move-rule draws decreased by 228.44%, and the average amount of moves per game decreased 35.14% from 1-step to 2-step. Overall, the results show notable improvement from 1-step to 2-step, as seen from White's perspective.

Unfortunately, due to limitations on both time and computational power, it was impossible to test or feasibly run games with more than 2 steps. There are no stalemates, rook capture draws, or threefold repetition draws in the results because White was able to make moves based on its calculation of Black's future moves. For example, if the White Rook were to move next to the Black King, one of Black's potential moves would be capturing the White Rook. The "endgameValue" variable inside of the position evaluator would be equal to 1000 for such a move, which would make this move highly undesirable for White and it would stand no chance of being chosen during White's minimizer step, since it would surely be one of the largest values in the subtree.

Despite not being able to run the game on more steps than 2, a hypothesis can be formed from the obtained results. An observation is that the number of games resulting in checkmate increases and the number of games resulting in 50-move-rule draws decreases, and that the average number of moves for the games decreases as well, when comparing the 2-step Minimax/Maximin games with the 1-step Minimax/Maximin games. This is because when the White King is n squares away from the Black King, n being the number of steps for the Minimax algorithm, it can assist the White Rook better in delivering the checkmate, by providing protection for the Rook. Conversely, the Black King would attack the White Rook whenever it would be n squares away from it, because it would be able to detect the value of 1000, awarded for a position where the White Rook is captured, n moves away.

Our hypothesis is that if the game were to run on 7 steps of Minimax/Maximin, both White and Black would play optimally, because their prediction and calculations would be within the entirety of the board's 8x8 range and each King would be able to draw closer to its piece of interest (the White King would come closer to the Black King, so as to assist in the checkmate and protect the White Rook from capture as both pieces drive the Black King towards the edge of the board, whereas the Black King would come closer to the White Rook, so as to capture it and incur a rook capture draw). The hypothesis also states that the 7-step Minimax/Maximin would result in a 100% chance of checkmate, so no games would go to 50 moves, and the average amount of moves would decrease as well, since both White and Black would play optimally. Therefore, the hereby proposed hypothesis is called The 7-Step Hypothesis.

V. Conclusion and Future Work

This paper presents a program that depicts a certain chess endgame scenario, namely White King and White Rook versus a Black King, both of which are played by an AI. It also hypothesizes that as the number of steps increase, the Minimax algorithm improves the AIs level of chess play and is capable of ending every game with a checkmate. This claim stems from an observation made from exhaustive testing and corresponding results obtained.

In order for further work to be done and to fully ascertain the capabilities of the Minimax algorithm, alpha-beta pruning may need to be incorporated to improve performance and lessen the processing workload. Because more computational power is needed, and because the amount of calculations that need to be done increases exponentially with every additional step, it is possible that a supercomputer will be needed to test the hypothesis, so that exhaustive testing could be done in a reasonable amount of time.

Acknowledgment

We would like to give our gratitude to Toni Jankuloski for allowing us to use his personal computer for the sake of the exhaustive testing and attaining the results demonstrated in this paper.

References

[1] Tesla.com. 2021. *Autopilot*. [online] Available at: <https://www.tesla.com/autopilot> [Accessed 17 August 2021].

[2] S. Russell and P. Norvig, 2003. Artificial Intelligence: A Modern Approach. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, p.55.

[3] S. Božinovski, 2016. Cognitive and Emotive Robotics: Artificial Brain Computing Cognitive Actions and Emotive Evaluations, Since 1981. In: ICT Innovations Conference 2016. Skopje, p.11.

[4] F. Jankuloski and A. Božinovski, 2020. Chess as Played by Artificial Intelligence. In: 12th ICT Innovations Conference 2020. Skopje: CCIS.

[5] F. Jankuloski, 2022. Signal Processing in an Artificially Intelligent Chess Program. Seminar paper for the Signal Processing course.. School of Computer Science and Information Technology, University American College Skopje.

[6] Simplilearn.com. 2022. *A* Algorithm Concepts and Implementation*. [online] Available at: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm> [Accessed 13 June 2022].