Bucknell University

# CHESS AI: A LOOK INTO AN ARTIFICIALLY INTELLIGENT CHESS AGENT

Final Project Report

Alex Godziela, Henry Kwan, Elias Strizower
CSCI 379 AI & Cognitive Science
Professor Chris Dancy

## INTRODUCTION

Artificial Intelligence (AI) is a field of study that looks into intelligence exhibited by machines. Throughout our class, we have learned about the psychology behind AI and what it means for a robot to be intelligent. In order to explore AI a little bit deeper, we decided to build a Chess AI for our final project--an intelligent computer that can optimize its moves in real-time to win a chess game against a human player. We chose to develop an AI specifically for Chess because Chess is a perfect information game that is particularly well-suited to intelligence analysis. To elaborate on that point, Chess is an extremely complex mathematical game; the game employs an endless amount of strategy and depth. In order for a player to win, one needs to strategically plan as the opponent makes each move.

## BACKGROUND & MOTIVATION

For those who may not be as familiar with the board game, Chess is a two-player perfect information game played on an 8x8 grid. Each player begins with 16 pieces: 8 pawns, 2 knights, 2 bishops, 2 rooks, 1 queen, and 1 king. These pieces are set up at opposite sides of the board and the goal is to place the opponent's king under checkmate, meaning that it is unable to get itself to a safe place where it is not being attacked. In certain cases, a game of chess will end in a draw, also known as a stalemate, where neither player is able to checkmate the opponent. With that being said, the goal of our Chess AI is to win games; it will only draw if necessary.

Our main goal for this final project was to explore intelligent machines and because of the wide array of logic that is used in Chess, we can actively determine the intelligence of our AI by analyzing the moves it is making. As stated above, Chess is a mathematical game that heavily

employs strategy. What this means is that a player needs to find a comfortable balance of attacking the opponent with the ultimate goal of checkmate as well as defending its own pieces to prevent one's own king from being cornered. In order to get to checkmate, each move needs to be well calculated. Since each chess piece has a given point value, we can easily differentiate good moves from those that are bad and it paves the way for our intelligence analysis.

**PERFECT INFORMATION**

Perfect information games are useful environments to study many of the different algorithms and decision making techniques that are used in AI cognitive science. We have already talked about many of these kinds of games in class, such as Go, and we have discussed how the various algorithms and techniques we have studied could contribute to AI made for playing these games. We would like to expand on what we have talked about in class and use the perfect information game of chess as the basis from which to study various types of AI related algorithms. Specifically, our goal is to create a chess AI that can beat a human opponent. We plan to implement this AI in a Python environment. The specific implementation of our AI remains to be determined. Our plan for this project will involve investigating the different approaches to making a chess AI, implement our own approach based on our research, and analyze the effectiveness of our implementation compared to other known chess AI implementations.

This is important because it demonstrates the capability of implementing a thinking system into a basic game. This paper will go into detail about the work that was done to accomplish the goal of creating an intelligent chess agent.

**BACKGROUND ON OUR CHESS AI**

The structure of our Chess AI was extremely important in order to build an effective chess agent. In order of evaluation, the chess AI must know the moves that it can make, the moves the opponent can make depending on its own moves, and then use heuristics and algorithms to determine which is the best move. The difficulty of making a good chess AI is that it has to know how to strategize and make moves similar to that of an advanced human player. When we think of how chess masters (people who play at the highest level of chess) play, they use multiple angles of attack and constantly strategize 5-6 moves ahead in order to win. However, for an AI to be able to do that, we have to write the heuristics so that it has logic to play the game and then algorithms so that it can strategize ahead and hopefully win against its opponent.

The main difference between a chess AI and a human playing the game is that the AI needs a set of rules that evaluate to a numerical value that it can rely on to make intelligent moves, whereas a human has the ability to think and reason and can apply their knowledge to choosing the move they think is best. In addition, given the quickly changing nature of a chess game, and intelligent chess agent must have the ability to determine its best move every turn. It may also have the ability to look more than one ply deep, and make moves that a human player may not see.

**COMMON CHESS AI ALGORITHMS**

Computers or robots with artificial intelligence employ many different algorithms, including, but not limited to, learning algorithms and search algorithms. For our Chess AI, we chose a search algorithm and will go into detail in the next few paragraphs to show the thinking process behind our decision.

Learning algorithms are a common implementation of artificial intelligence in a computer. These algorithms take input from their environment, and use it to train their model and produce an output. Andres Munoz once described machine learning as giving "computers the ability without being explicitly programmed." Once the inputs are put into the learning model, they are cross referenced with the expected output, and the model changed accordingly to try to output closer to the expected result. The model is then trained with various inputs and expected outputs until it has "learned" how to perform the calculations correctly.

Another type of algorithm used in artificial intelligence is a search algorithm. These algorithms tend to utilize a cause-and-effect concept where the search considers each possible action available at a given moment. These algorithms, combined with a set of rules, evaluate which decision is most intelligent. The different between search algorithms and learning algorithms is that search algorithms do not change their model over time.

These types of models can be used to build an intelligent chess agent. In particular, search algorithms fit very well with chess since they evaluate and return an intelligent decision considering each possible action at a given moment. Given chess, this means the algorithm will consider all possible moves at a given turn and decide which of the moves is the best. Coupled with a learning algorithm, it can evaluate if the moves are truly intelligent, and change its model accordingly over time.

**MINIMAX ALGORITHM & HEURISTICS**

In able to accomplish the goals of our project, we believe that basic heuristics in combination with the minimax algorithm will successfully implement an intelligent chess agent. The role of the heuristics is to assign a numerical value to each of the chess pieces and

subsequently each chess move has an associated point value. The way the minimax algorithm works is that we know each of the point values increases from pawn to queen, and the minimax algorithm maximizes the point value of the AI while minimizing the point value of the opponent based on the AI's previous move.

We chose this approach because it fits best with the manner in which chess is played. Specifically, having the minimax algorithm evaluate the best move at each turn should work well with the game of chess given the dynamic nature of the game. Although other approaches were considered, such as deep learning and reinforcement learning, we believed that it would be less effective to train a model on the best move since chess changes drastically each turn. This is why we believed that it was best to implement a model that could take any board state and return the best move solely based on that.

Minimax is very well known algorithm that is used in many different contexts. It is known best for application in two-player, zero-sum games. The algorithm is also used in game theory, a branch of mathematics used to evaluate competitive situations between two players. It is commonly used in business, finance, and economics, and relies on minimax to determine the best possible outcome. The aim of minimax in financial and business applications is to cause "the least amount of regret, should the strategy fail."

Heuristics are also very popular and used in many contexts. A heuristic is viewed as a rule of thumb, or method of granting a short decision-making time based on a set of rules. They not only apply to the field of computer science, but to psychology as well. In terms of computer science, heuristics are not only used for algorithms, but for user interface design as well. User interface design relies heavily on 10 heuristics that were developed by Jakob Nielson, such as

error prevention, user control and freedom, among others. In general, heuristics are very commonly used as rules of thumb to quickly gaining intelligence on a situation.

Minimax can be clearly understood by the image below:

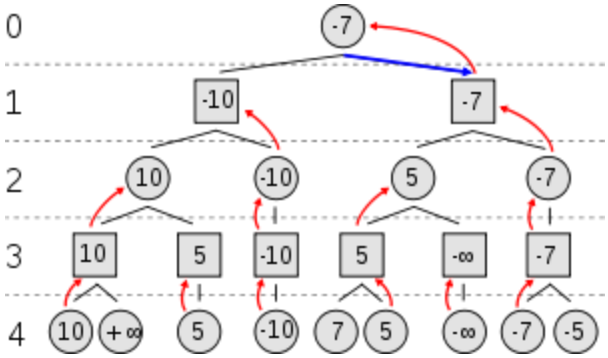


Figure 1: Visualization of Minimax

Looking at the figure above, minimax works by assigning alternating tree depths to moves between the player and the AI. For example, level 0 refers to the first move the AI would make and level 1 shows all the moves the player could make depending on the the AI's first move. The diagram is simplified for simplicity and in real life chess games, level 1 would have many more moves and the tree would be much, much larger. After this, depending on the chosen ply or depth in the minimax algorithm, we will calculate points by looking at the root nodes and calculating a difference between the total points the AI accumulated versus those of the user. The highest positive number indicates the best move the AI can make whereas the most negative number indicates the worst move for the AI and the best move for the opponent. After finding the best move for the AI, we propagate that value back up the tree and pass that as our best move.

Both minimax and heuristics were informed by psychological theories. Minimax has its base in social psychology, specifically the theory of social situations. The theory of social situations deals with the tradeoff of maximizing and minimizing between two individuals. The

relationship between this theory and minimax is prevalent in the exchange between two agents, and their attempt to maximize for themselves, and minimize the other. Heuristics is also directly related to psychology, in that they are used in decision making and problem-solving. They are the way in which people quickly evaluate situations, though they may not necessarily be conscious of it. As pointed out by Kendra Cherry in her article "What Is a Heuristic and How Does It Work?", although heuristics are necessary for decision making, they can also lead to cognitive bias. She explains that "just because something has worked in the past does not mean that it will work again, and relying on an existing heuristic can make it difficult to see alternative solutions or come up with new ideas." This not only applies to heuristics in a psychological setting, but also to computer science. If heuristics are implemented in a program, it is important to realize that just because they worked before does not mean they will always be as effective. It was important to take this into account when designing our heuristics for the chess AI, because we had to make them general enough to apply to any game of chess, but specific enough to be effective.

**CHESSNUT**

For this project we decided to use a Github project called Chessnut, developed by Chris Gearhart et al. as well as a basic graphical user interface (GUI). We decided to work with Chessnut because it provides a simple python chess model, and stores the state of the game in Forsyth Edwards Notation (FEN). FEN was important to the implementation of this project because it made it simple to parse the FEN string into our GUI. In addition, we chose Chessnut because it is not an engine and lacks its own GUI, so it did not achieve any of the goals we were trying to accomplish with our agent. The library's clean code and useful methods set a solid

stage for us to integrate our AI chess agent and GUI. Lastly, in order to focus the majority of our efforts on the AI, we settled on a simple GUI encoded in American Standard Code for Information Interchange (ASCII), which models an 8x8 chess board, and represents pieces with letters, designating color by the letter's case.

**PROJECT PLANNING**

Our project was planned in predetermined iterations to simplify keeping track of progress and integrating new functionality. The first step was to create a GUI that worked with Chessnut to completely model a game of chess, taking user input. The next iteration was to build in simple AI. This AI chose randomly from a list of its next possible moves. Although this iteration was very simple, we believed it was an important landmark to make certain that we could build in a chess agent with Chessnut that could play against a human. After this was done, our next iteration was to implement heuristics into the agent so it could intelligently choose its next move.

The first heuristic we implemented was ranking moves by the point value of the pieces. At the start of the game, many, if not all, of the spaces in rows 3-6 of the chess board will be empty and those point values will be 0. However, once the game has gone into full swing, a pawn that has a possible move of taking a knight will be assigned a point value of 3 points.
This demonstrated that our AI had the capability of being intelligent, though basic, and pose a slight challenge to the human.

The last iteration was the most important for us to implement, though most difficult, which was to successfully integrate the minimax algorithm into our chess AI. This posed a huge challenge and took a lot of work to complete. After completion,though, our chess agent is very intelligent and provides fair competition to the human.

## GROUPWORK

We divided our group work by looking at each of our team members' strengths and worked on different parts accordingly. Henry worked on coding the minimax algorithm and heuristics, building the node class, writing helper functions for the minimax algorithm such as determining what move was selected by minimax, and writing the readme file in markdown. Alex's strength was writing and put together an in-depth outline for our final report and helped Henry with the minimax algorithm. Elias was tasked with implementing the GUI and integrating it with Chessnut, writing a function to run the game, building in the ability to look ahead 2 ply, and helping implement minimax.

Given that our group was a disadvantage and lacking a fourth person, we tried our best to distribute tasks and outline the progression of the project. Elias worked with Chessnut because he felt most comfortable reading through their API. Given his understanding of Chessnut, he helped the group work through all of the problems that we encountered with the library. Henry felt comfortable learning about the minimax algorithm and thus he wrote and implemented it into the code base. Alex voiced his strengths with writing and took charge of the outline, which led to the start of our paper.

Overall, we all helped each other out when we got stuck and worked as a cohesive group to move the project forward effectively. Decisions were always made as a group. The scope and outline of our iterations were decided as a group, and we monitored the rate at which work was completed, as to not fall behind.

## RESULTS

We successfully created a chess agent that has the ability to intelligently choose its moves. Using different heuristics and algorithms, such as minimax, the chess agent can intelligently choose from its possible moves which is the best given the current state of the board.

Tests were done on the runtime and piece capturing ability of three algorithms, 1 Ply Random Move, 1 Ply Best Move, and 2 Ply Minimax. The results of the tests are below:

| Move Duration of Different Algorithms (seconds) | | | |
|---|---|---|---|
| | 1 Ply Random Move | 1 Ply Best Move | 2 Ply Minimax |
| Turn 1 Time | 9.060E-06 | 1.335E-02 | 5.752 |
| Turn 2 Time | 6.914E-06 | 1.535E-02 | 4.956 |
| Turn 3 Time | 7.868E-06 | 1.636E-02 | 16.383 |
| Turn 4 Time | 7.868E-06 | 1.656E-02 | 14.788 |
| Turn 5 Time | 8.821E-06 | 1.754E-02 | 16.234 |
| Turn 6 Time | 7.153E-06 | 1.622E-02 | 12.114 |
| Turn 7 Time | 8.106E-06 | 1.668E-02 | 15.100 |
| Turn 8 Time | 7.153E-06 | 1.512E-02 | 15.096 |
| Turn 9 Time | 8.106E-06 | 1.633E-02 | 16.190 |
| Turn 10 Time | 8.106E-06 | 1.528E-02 | 16.327 |
| **Average Time** | 7.915E-06 | 1.588E-02 | 13.294 |

Figure 2: Move Duration of Different Algorithms

The table above displays how long it takes the particular algorithms to make their moves. It can be seen that 1 Ply Random Move is the quickest, followed by 1 Ply Best Move, and 2 Ply Minimax. It should be noted that 2 Ply Minimax is significantly slower than the two other algorithms because it has to do many more computations to look two moves ahead for every possible chess move. 2 Ply Minimax is roughly 1.6 million times slower than 1 Ply Random Move, and 800 times slower than 1 Ply Best Move.

| Game Score of First 20 Moves (points) | Game 1 | Game 2 | Game 3 |
|---|---|---|---|
| **2 Ply Minimax vs 1 Ply Random Move** | | | |
| 2 Ply Minimax Score | 14 | 21 | 23 |
| 1 Ply Random Move Score | 5 | 0 | 9 |
| **2 Ply Minimax vs 1 Ply Best Move** | | | |
| 2 Ply Minimax Score | 22 | 23 | 12 |
| 1 Ply Best Move Score | 9 | 15 | 9 |
| **1 Ply Random Move vs 1 Ply Best Move** | | | |
| 1 Ply Random Move Score | 3 | 1 | 16 |
| 1 Ply Best Move Score | 10 | 22 | 18 |

Figure 3: Game Score of First 20 Moves

In a series of three games, the scores of different algorithms playing each other can be seen in the table over. In both games that 2 Ply Minimax play it consistently outperformed the other algorithms. Also, it can be seen that 1 Ply Best Move performed better than 1 Ply Random Move in every case as well. From this test, 2 Ply Minimax is the best, followed by 1 Ply Best Move, and 1 Ply Random Move being the worst.

Overall, in 2 Ply Minimax is the best algorithm because, although it is computationally slower, it delivers a significantly higher score than the other two algorithms. For each AI move, it looks two moves ahead to calculate a point differential and that is the key to continually having a point advantage over the two other agents which only look at the next best move. The logic behind minimax is made to be better than any other agent--since the ply/depth directly corresponds with the number of moves it looks ahead. Thus, if it had a depth of 6, it would probably be able to beat most human players since people cannot do the computations for all possible moves six turns down the road all within memory. The only downside to minimax is that as depth increases, performance decreases--computation time increases exponentially with each additional depth.

One way to combat this is by adding Alpha-Beta pruning, which our team did not have time to implement. Pruning is harder to implement with chess compared to simpler two player games because our point value differential and list of possible moves is much larger and more complicated than typical games. An important thing to remember is that chess has six different pieces that all move in unique patterns. For example, a similar game, such as checkers, has one unique piece with one moving pattern.

- Propose some process (that we would outline in this section) to solve our problem
  - Use neural nets to simulate learning (for example)
    - combine with an effective and well-known chess AI algorithm (learning on top of an AI that can already play chess)
  - Base this off of the research we conducted
    - Why did we choose this particular approach
      - Why DIDN'T we choose a different approach?
    - How is it related to AI/Cog science
  - Must discuss how our approach is used in other environments
  - Give a broad overview
    - Break down the major components
    - Give a diagram/visualization (he wants this, so we should make one even if there are none we can cite from the internet)
  - Was this informed by and particular psychological theory
  - What will we be working with
    - specifically what python chess API we will be using along with any chess visualization techniques

**Progression of the project**

- Must answer

- How was our project planned? how was the work divided up and executed by team members? how did we utilize each other's' strengths and weaknesses? How was communication conducted? How were decisions made?
    - Discuss our repository breakdown/commit strategy here as well

**Demonstration**

- How are we showing the results of our project?
    - Provide here the detailed plan of how we are presenting our findings/collecting data on the success of our chess AI

ALEX THE REST IS FOR YOU:

- The two sub-headers below for this paper (Results and Conclusion portion… although results was started, just continue it)
- APA format for this paper (bibliography, and figures)
- After, make sure everything has consistent formatting (TNW 12, indents, justified, etc)
- The user manual
- Text in groupme after you're finished? Thx

Bibliography


https://www.chess.com/learn-how-to-play-chess
https://medium.freecodecamp.com/simple-chess-ai-step-by-step-1d55a9266977
Machine Learning and Optimization - Andres Munoz
http://www-cs-faculty.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html
http://www.businessdictionary.com/definition/minimax-principle.html
https://neos-guide.org/content/game-theory-basics
https://www.verywell.com/what-is-a-heuristic-2795235
https://tfa.stanford.edu/download/TenUsabilityHeuristics.pdf
http://users.sussex.ac.uk/~christ/crs/kr-ist/lec05a.html