FAI Draft Report

Mason Leon and Naveen Kumar Chiluka

CS 5100
Foundations of Artificial Intelligence
Dr. Lawson L.S. Wong
Fall 2019

Khoury College of Computer Sciences
Northeastern University
Boston,MA

Contents

1 Introduction

1.1 Problem

Chess has been a challenging game to be solved for many years in the field of AI. It's been tough due to the fact that its state space is too large (around 10^{50} states) and it would be computationally impossible to play a game without any approximation. In order to build a good AI agent there need to be heuristics for every move to finish the game.

1.2 Motivation

The game provides a good opportunity to learn about designing evaluation functions based on the features of the game. It also helps us in understanding the nuances of the game through an AI lens and apply the concepts learned from the course to another game.

1.3 Objective

To build an AI Chess Agent using Minimax Algorithm, Alpha-beta pruning along with a reasonable evaluation function.

1.4 Problem formulation

- State space: All the valid board configurations.
- Action space: All the valid moves of the pieces.
- Goal state: The state where the AI agent wins, loses or draws.
- Utility or Evaluation function: Heuristic based on key features.

1.5 Ideal Outcome

Build an AI agent using different evaluation functions and analyze the performances of each of these cases.

2 Approach

2.1 Environment

We initially planned to use Javascript environment for the project but as both of us are completely new to front-end development it was taking longer than expected to comfortably code in JS. As we researched implementation of our algorithms, we discovered python-chess, a pure Python chess library with move generation, move validation and support for common chess formats. This project was exciting for us because it allowed us to work on the project in a language we were comfortable with as well as utilize data science tools such as Jupyter Notebooks, Google Colab, and Pandas for running, benchmarking, and performing statistical analysis of our results. Additionally, this library allows for integration of popular chess engines for the purpose of testing our algorithm against another intelligent chess player agent.In reading the python-chess documentation, we were directed to a Jupyter Notebook from Dr. Douglas Blank' CS371: Introduction to Cognitive Science course at Bryn Mawr College. In this starter code we learned about the library's basic Board, Game, and Piece data structures and state representation of the library. We adapted the examples of a random agent, a human agent, naive agent, and their respective evaluation functions for our own project and have set up the chess game environment.

2.2 Agents

2.2.1 Random Agent

We have started with a Random Evaluation function which picks a random move from the available legal moves. As expected, the results were random.

2.2.2 Naive Agent

So, after doing some research on the chess evaluation strategies, we discovered Material scores for the board. Material score is a heuristic that computes the utility of a particular state of the board based on the available pieces and the corresponding weights. By assigning weights to different pieces, the following objectives can be achieved:

- We can take down the
- A disadvantage exchange like exchanging a minor piece for a major piece can be avoided.
-

$$MaterialScore = \sum_{\forall\, i\, in\, pieces} w_p(i) * (nW(i) - nB(i))$$

where
i = piece type
nW(i) = number of white pieces of type i
nB(i) = number of black pieces of type i

Based on the reasoning provided on the Chess Programming Wiki we decided on the following values for the weights for our material score heuristic:
Pawn = 100
Knight = 320
Bishop = 330
Rook = 500
Queen = 900

2.2.3 Improved Naive Agent

We found that the Material Score heuristic is not of much help since its effect is seen only when there is a capture. For that to happen the agent's pieces should be in positions where it's possible to capture the opponent's pieces. We observed that the Naive agent was resulting in draws when pitted against a Random Agent more often than not. So, we imparted a little more intelligence to it by adding a couple of additional features like:

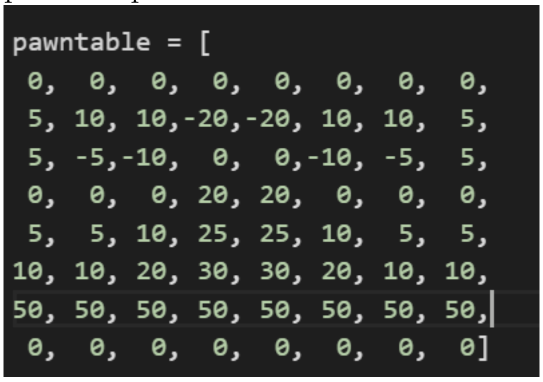
- score would be increased by 10 if the move results in a capture.
- score would be increased by 1000 if the move results in a checkmate.
- score would be increased by 100 if the move results in a check.

- score starts from a random seed unlike in the previous evaluation where it started from zero.

Adding these features, showed some positive results in the performance of the agent and resulting in wins most of the time against the Random Agent.

2.2.4 Advanced Agent

The Improved Naive Agent was a reasonable player and picked better moves over Naive agent but it would still end up picking sub-optimal moves since the features are not enough. In search of a better heuristic we found *Piece-Square tables.* The tables can be thought of as Reward functions for each of type of piece. There would be positive and negative rewards for each square of the board for every type of piece. So, we created an Advanced Agent which relies on both Material Score and Piece-Square Tables. Please refer to section 6 for the actual tables used in the project. This agent performed far better than its predecessor against the Random Agent.
To illustrate the concept of Piece-Square Tables, consider the following Pawn Table:
In the below table, the top rows represent the white side while the bottom two rows represent the black side and the table is used for white pieces.The positive values are rewards for the agent if a pawn is moved to that square while negative values are penalties for moving to that square. This can dictate the movement of pawns in the intended directions. As per the table, the central pawns are levied with heavy penalty if they don't move forward and make room for minor pieces and queen. All the pawns are encouraged to make it to the other end so that they can be promoted to more powerful pieces.



```
pawntable = [
 0,  0,  0,  0,  0,  0,  0,  0,
 5, 10, 10,-20,-20, 10, 10,  5,
 5, -5,-10,  0,  0,-10, -5,  5,
 0,  0,  0, 20, 20,  0,  0,  0,
 5,  5, 10, 25, 25, 10,  5,  5,
10, 10, 20, 30, 30, 20, 10, 10,
50, 50, 50, 50, 50, 50, 50, 50,
 0,  0,  0,  0,  0,  0,  0,  0]
```

- Similarly, the Knights are encouraged to go to the center as they have the highest degree of freedom there.
- The Bishops are discouraged from moving to corners and borders. They are encouraged to move to the center as they serve better purpose there.
- The Rooks are discouraged from moving to the edges as they are underutilized there. They are incentivized either to go to the opponent's side or not-penalized for being in the central square.

2.2.5 Mini-max Agent

Just the two heuristics were not enough for our agent to perform well against other intelligent chess agents. The agent needed to look ahead into the future and predict the moves of the opponent and then make decisions. So, we implemented a mini-max search algorithm along with the Material Score and Piece-Square Table heuristics. This was by far the best agent we have created so far.

2.2.6 Mini-max-Alpha-Beta Agent

Although the Mini-max Agent was faring well with other agents, it was taking too long to make a move when the depth of the search was more than 3. So to tackle this problem, we improved the

Mini-max algorithm by incorporating the Alpha-Beta pruning technique. This technique will help the agent avoid expanding unnecessary moves.

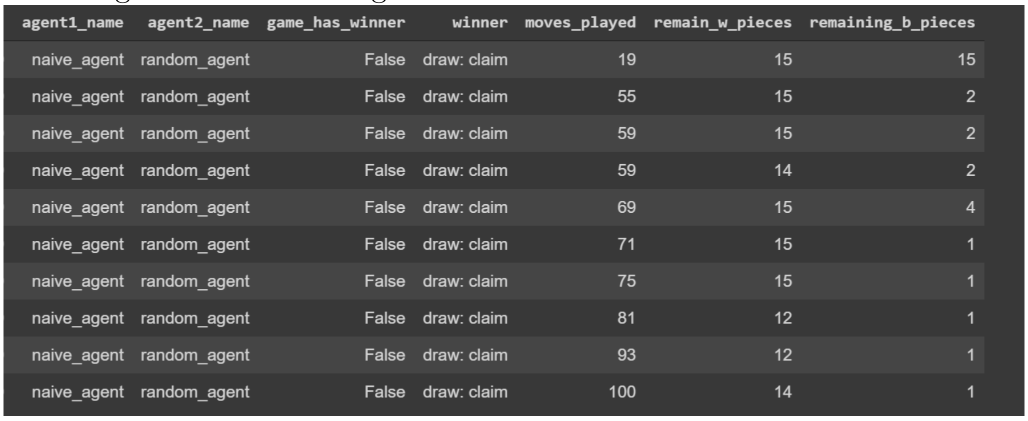
2.3 Tools/Libraries

Development Environment	Libraries
Jupyter,Pycharm	python-chess
Anaconda	stockfish
Pandas	Syzygy API

3 Results

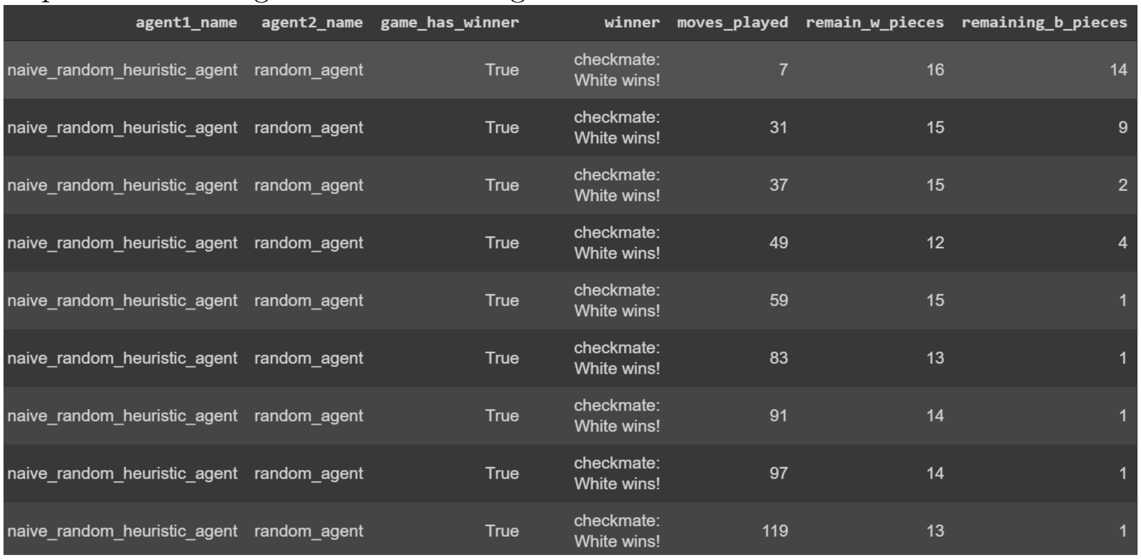
Mini-max-Alpha-Beta Agent is now improved due to the Advanced Evaluation Function in the following ways:

- Number of moves required to win
- Reduced overall losses (leading either to a win or a draw)
- Naive Agent vs Random Agent



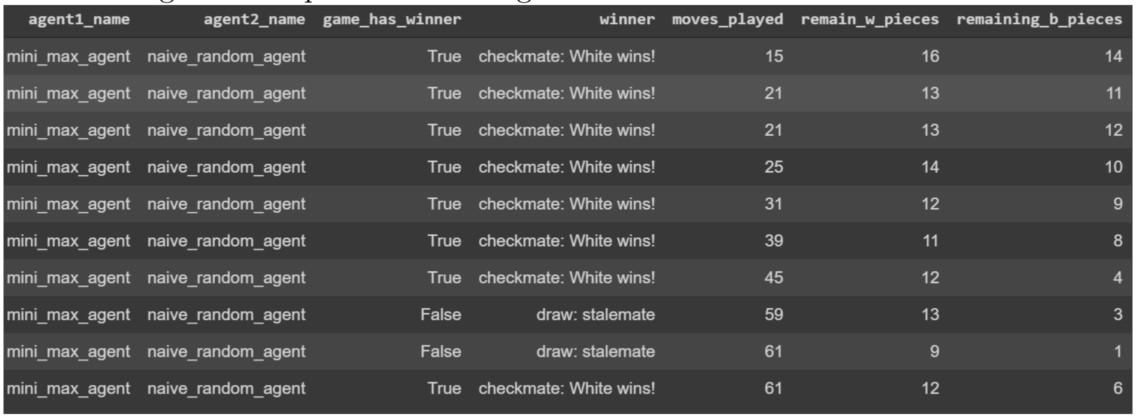
agent1_name	agent2_name	game_has_winner	winner	moves_played	remain_w_pieces	remaining_b_pieces
naive_agent	random_agent	False	draw: claim	19	15	15
naive_agent	random_agent	False	draw: claim	55	15	2
naive_agent	random_agent	False	draw: claim	59	15	2
naive_agent	random_agent	False	draw: claim	59	14	2
naive_agent	random_agent	False	draw: claim	69	15	4
naive_agent	random_agent	False	draw: claim	71	15	1
naive_agent	random_agent	False	draw: claim	75	15	1
naive_agent	random_agent	False	draw: claim	81	12	1
naive_agent	random_agent	False	draw: claim	93	12	1
naive_agent	random_agent	False	draw: claim	100	14	1

- Improved-Naive Agent vs Random Agent



agent1_name	agent2_name	game_has_winner	winner	moves_played	remain_w_pieces	remaining_b_pieces
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	7	16	14
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	31	15	9
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	37	15	2
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	49	12	4
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	59	15	1
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	83	13	1
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	91	14	1
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	97	14	1
naive_random_heuristic_agent	random_agent	True	checkmate: White wins!	119	13	1

- Minimax Agent vs Improved-Naive Agent



agent1_name	agent2_name	game_has_winner	winner	moves_played	remain_w_pieces	remaining_b_pieces
mini_max_agent	naive_random_agent	True	checkmate: White wins!	15	16	14
mini_max_agent	naive_random_agent	True	checkmate: White wins!	21	13	11
mini_max_agent	naive_random_agent	True	checkmate: White wins!	21	13	12
mini_max_agent	naive_random_agent	True	checkmate: White wins!	25	14	10
mini_max_agent	naive_random_agent	True	checkmate: White wins!	31	12	9
mini_max_agent	naive_random_agent	True	checkmate: White wins!	39	11	8
mini_max_agent	naive_random_agent	True	checkmate: White wins!	45	12	4
mini_max_agent	naive_random_agent	False	draw: stalemate	59	13	3
mini_max_agent	naive_random_agent	False	draw: stalemate	61	9	1
mini_max_agent	naive_random_agent	True	checkmate: White wins!	61	12	6

4 Problems Faced

- Analyzing the game and deciding on the features was tougher than we thought.
- Due to the space and time complexity of the game, it was difficult to debug the game as every game would take long time to run till the end depending on the evaluation function. We could use PyCharm for debugging but it was not a feasible option for us due to the limitations of our local machines. So, our only option was to run on google co-lab servers but the debugging features of the Jupyter notebook were limited and not very helpful.
- We encountered difficulties while integrating our agent with another trained, intelligent chess agent like Stockfish. The problem was the Jupyter support with File IO. We had Permission errors while uploading the executable file of the Stockfish engine. We are still in the process of fixing this issue.
- There were confusions over obscure chess rules automatically built into the library resulting in draws.

5 Next steps

- We are currently working on finishing migrating utility and API functions to an external Python file to ease Jupyter notebook debugging.
- We intend to integrate a trained, external agent (stockfish) with our program to see our agents fare with that.
- We plan to utilize an API with endgame statistics to compare the performance of our agents against millions of others at same state and use the data to determine if our heuristics are improving performance.

6 References

- Fiekas, Niklas et al. python-chess: a pure Python chess library. Github. `https://github.com/niklasf/python-chess`
- Fiekas, Niklas et al. python-chess: a pure Python chess library. ReadtheDocs. `https://python-chess.readthedocs.io/en/latest/index.html`

- Chess Strategies https://en.wikipedia.org/wiki/Chess_strategy
- https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/
- https://static.aminer.org/pdf/PDF/000/226/325/genetically_programmed_strategies_for_chess_endgame.pdf
- Piece-Square tables and Material Score https://www.chessprogramming.org/Simplified_Evaluation_Function