ATOMICAI, A DEEP LEARNING APPROACH FOR ATOMIC CHESS HEURISTIC

Ismail Ouazzani Chahdi
Student# 1006804095
ismail.ouazzani@mail.utoronto.ca

Anay Shah
Student# 1006975472
anay.shah@mail.utoronto.ca

Eric Lefort
Student# 1006896245
eric.lefort@mail.utoronto.ca

1 INTRODUCTION

The high-level objective of this project is to create an artificial intelligence system that is capable of playing the chess variant *Atomic Chess*. This project focuses on the heuristic function, which, given a board position, will evaluate the strength of the board position, from the perspective of ”white”. The model will be a deep neural network, which takes as input a special grid representation of a chess board and returns a value between 0 and 1, with values closer to 1 indicating stronger positions for white and values closer to 0 indicating stronger positions for black. See Table 3 for input format.

We will integrate this heuristic function with a search-based chess engine, to allow the AI to play against human players and other AIs. Our github can be found here.

There are a few reasons that deep learning is a good approach to this problem. Chess is a deterministic, complete-information game, where the actual strength of a position should only reflect the likelihood of white winning the game, however, this outcome is both volatile and far removed from the current state, making it difficult to determine. By combining the board features using a complex model where the weights are learned empirically from large datasets of games between skilled players, we hope to be able to capture the key predictors of game results. Additionally, deep learning has been effectively applied to the equivalent problem in both classic chess and the crazyhouse variant of chess.

2 ILLUSTRATION / FIGURE

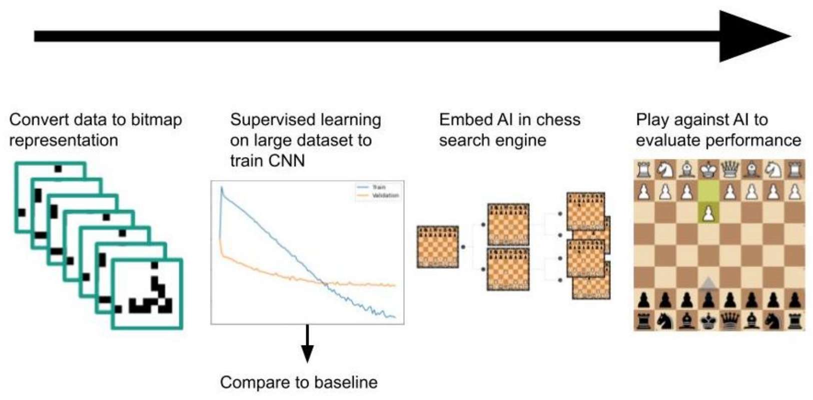

Convert data to bitmap representation
Supervised learning on large dataset to train CNN
Train
Validation
Embed AI in chess search engine
Play against AI to evaluate performance
Compare to baseline

Figure 1: High-level diagram of model development steps

3 Background

3.1 Chess Engine Overview

Chess-playing artificial intelligence exists since the 1990s with the development of Deep Blue at IBM, the supercomputer system that went on to defeat Garry Kasparov, the then-world-champion in 1997. We focus on chess engines implemented purely software, which work in a very similar manner to Deep Blue and other early computer systems designed for this purpose. Campbell et al. (2002)

3.2 Search engine

Given a chess game, on its turn, (whether that is black or white) the AI will consider the current board state and the possible moves it can make, which produces the successor board states. By recursively expanding these successor states, the AI forms a search tree of possible future states, from which it must select the optimal path forward, which then determines the required move. The engine performs a version of min-max search, with many possible optimizations. Campbell et al. (2002)

In order to focus on building the heuristic, we decided not to implement the search engine ourselves to give us more time to improve the evaluation function. We identified a simple search-based chess engines which would allow for relatively easy modification: Andoma developed by Andrew Healey using python-chess. Healey (2020) Being written in Python, the engine allowed us to seamlessly integrate our model as we could make use of the PyTorch API.

There also exist other chess engines that we identified as being options for the framework in which to use our neural network. The main one of these was written in C: Tom Kerrigan's Simple Chess program, which implements a simple search algorithm with room for improvement. This program is advantageous compared to other open-source chess engines because it is very straightforward to modify the search algorithm and the evaluation function. Kerrigan

3.3 Evaluation Function

When the AI reaches the search depth, it relies on an evaluation function, which provides a scalar-valued estimate of the strength of a board position, without considering any successor states. For many chess engines this comes down to computing a weighted sum of a series of handcrafted features, such as remaining piece count, board position and centrality, pawn structure, piece mobility, etc. Campbell et al. (2002)

3.4 Crazy Ara

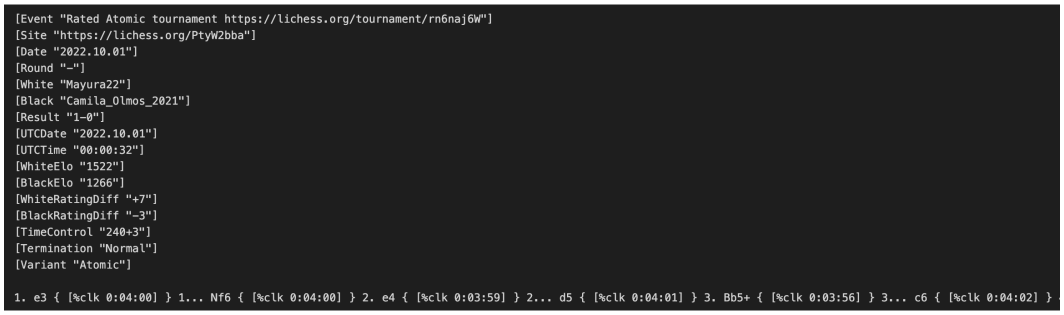
While most classical chess engines are designed with handcrafted evaluation functions, this is largely due to the existence of a large body of chess knowledge and expertise. Due to the much smaller number of active players, variants of chess do not have the same body of domain knowledge and knowledgeable experts to draw from in order to handcraft a good evaluation function.

As previously demonstrated by researchers, it is possible to use Deep Learning to play chess, go, and other games with no prior knowledge. Notably, a group of researchers from TU Darmstadt developed an effective deep neural network model to play the Crazyhouse variant of chess, above the world champion level. Czech et al. (2020). Their approach was based on google's Alpha Go AI, which combined Monte Carlo Tree Search with a heuristic and a policy, both updated by deep learning methods Silver et al. (2016).

4 Data Processing

We collected 900,000 games from the lichess variant database. We transform each board so that the current player is at the bottom (white). Since chess is a zero-sum game, flipping the board should just yield the negative of the evaluation in the non-flipped board. The chess pieces of both players are then transformed to a 17 channel binary-map of size 8x8 using the python-chess library which encodes the position of all pieces. An example input tensor of the model is displayed in Table 3.

The dataset consists of a set of PGN files, which contain all the games played on Lichess each month. Each game possesses headers, so we are able to filter out bad games by ignoring games without headers or games that were abandoned by one of the players. Figure 2



```
[Event "Rated Atomic tournament https://lichess.org/tournament/rn6naj6W"]
[Site "https://lichess.org/PtyW2bba"]
[Date "2022.10.01"]
[Round "-"]
[White "Mayura22"]
[Black "Camila_Olmos_2021"]
[Result "1-0"]
[UTCDate "2022.10.01"]
[UTCTime "00:00:32"]
[WhiteElo "1522"]
[BlackElo "1266"]
[WhiteRatingDiff "+7"]
[BlackRatingDiff "-3"]
[TimeControl "240+3"]
[Termination "Normal"]
[Variant "Atomic"]

1. e3 { [%clk 0:04:00] } 1... Nf6 { [%clk 0:04:00] } 2. e4 { [%clk 0:03:59] } 2... d5 { [%clk 0:04:01] } 3. Bb5+ { [%clk 0:03:56] } 3... c6 { [%clk 0:04:02] }
```

Figure 2: Representation of a Single Game in the Dataset

Each board is labelled with 1 if the current player won the game and 0 otherwise. We do this so that when our model sees new data, it's level of confidence in the winner becomes an estimate of how good a board is, which is precisely what we need for a heuristic function. To make the label more meaningful - i.e. players do not play randomly - we only kept games were both players had an elo rating greater than 2000 (chess experts). From these games, we were able to randomly sample 2,000,000 boards from the 900,000 games to reduce correlation introduced by choosing successive positions, as suggested by Silver et al. (2016). This allowed us to reduce the overfitting we encountered in the first versions of our model.

Out of the 2,000,000 boards sampled, we used 60% for training, 20% for validation and 20% for testing. More games (therefore boards) are available in the Lichess dataset for further testing with unseen data. However, for capacity limitations we were limited to 2,000,000 boards, as the dataset occupies 20GB of disk space.

4.1 Data Analysis

Having acquired the data files from Lichess, we unzipped them and processed them using the Python Chess Library. The resulting database from 300,000 files was first analyzed for quality and correctness; Each game was checked for missing headers or other form of information, and therefore the ones abandoned before any moves by either player were bypassed. Next, the data was subjected to the main analysis and the following statistics were measured:

- White to Black Win Ratio
- Players' Elo Distribution
- Game Duration Distribution (in terms of # of moves)
- Common Opening Moves

The white-black win ratio was found to be 54-43 in favour of the white player. The remaining portion constitutes draw results. The other statistics can be better represented in Figure 3 .

5 Architecture

We use a CNN with repeated sequences of Conv2D, ReLu, and max pooling layers with a linear layer to get the final output. A CNN is the appropriate choice of model because the strength of a board is given by the placement of pieces and their spatial relationships, which can be captured by the model. CNNs are often used to develop heuristic functions in board games for this property, as shown by Silver et al. (2016). The precise architecture of the model is described in table 1.

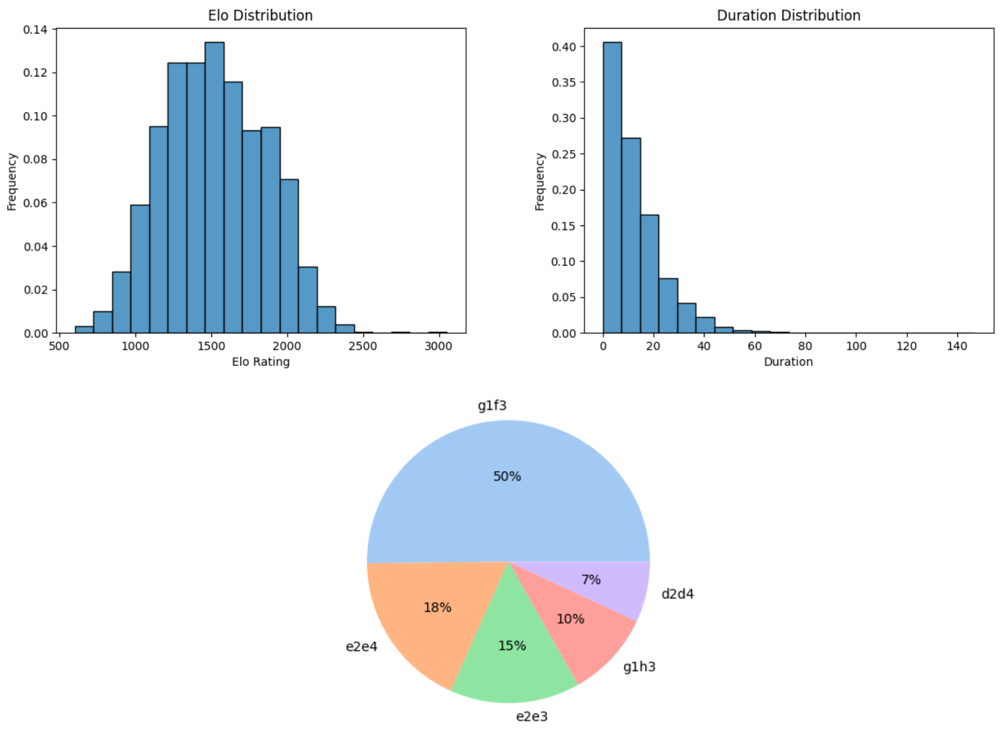


Figure 3: Players' Elo and Game Duration Distributions, and 5 Most Common Openings

Table 1: Model Summary

Layer (type)	Output Shape	Number of Params
Conv2d-1	[-1, 32, 8, 8]	4,928
ReLU-2	[-1, 32, 8, 8]	0
MaxPool2d-3	[-1, 32, 5, 5]	0
Conv2d-4	[-1, 64, 5, 5]	18,496
ReLU-5	[-1, 64, 5, 5]	0
MaxPool2d-6	[-1, 64, 3, 3]	0
Conv2d-7	[-1, 128, 3, 3]	73,856
ReLU-8	[-1, 128, 3, 3]	0
MaxPool2d-9	[-1, 128, 2, 2]	0
Conv2d-10	[-1, 256, 2, 2]	295,168
ReLU-11	[-1, 256, 2, 2]	0
MaxPool2d-12	[-1, 256, 2, 2]	0
Linear-13	[-1, 512]	524,800
ReLU-14	[-1, 512]	0
Linear-15	[-1, 128]	65,664
ReLU-16	[-1, 128]	0
Linear-17	[-1, 1]	129
Total params	983,041	

6 Baseline Model

The baseline model relies on simply evaluating straightforward chessboard features, including:

1. Material value (piece value count)
2. Mobility score (number of legal moves available)

3. King safety (Closeness of friendly pieces to enemy king, and vice-versa)

It currently has a sign accuracy (predicting who will win) of 60%, and an BCE Loss of 0.47. This baseline is a reasonable choice, since Material value, Mobility score, and King Safety are traditionally used in chess heuristic functions.

7 Quantitative Results

Our model performs well given the difficulty of the problem at hand, getting a classification accuracy of 0.625 with a BCE loss of 0.349 on the validation set. These resultes are significantly better than the ones obtained with the baseline model.

Table 2: Dataset Loss and Validation

Set	BCE Loss	Accuracy
Test	0.35	0.687
Validation	0.349	0.625
Train	0.247	0.834

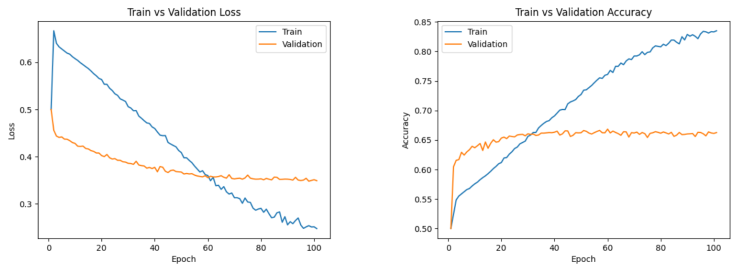


Figure 4: Model accuracy and loss, on the training and validation sets

Our model overfits significantly after 60 epochs, so we keep the model obtained at 60 epochs for playing and testing purposes.

8 Qualtitative Results

8.1 Game duration

In our dataset, the mean number of turns in a game was only 7.6 moves. In comparison, during our tests, our model played games lasting, on average, 10.2 moves over our 15 tests when playing both against advanced AIs (stockfish) or experienced human players (greater than 1600 elo). Managing to play such long games indicated that our model has a defensive play style, especially when playing Black.

8.2 Opening

We noticed that our AI prefers to open with e2e3, which is one of the most common openings in Atomic chess, indicating that our model is able to learn varient-specific tactics. Using this opening, it is able to defeat our baseline every time.

9 Evaluate on new data

To test our model on new data, we obtained its accuracy and validation in the testing set, as presented in table 2.

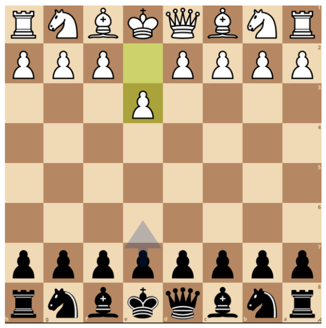


Figure 5: Opening of our Model starting at White

We also tested the model in another 500,000 boards from the Lichess database, and obtained an classifying accuracy of 0.67 with a BCE Loss of 0.35, indicating that our model performs well on new samples.

Furthermore, after integrating the AI with the engine, we conducted testing against other AIs such as Stockfish and Andoma's baseline heuristic and against human players. Our AI was able to consistently defeat Andoma with the default heuristic, and human players with elo smaller than 1200. However, our AI could not defeat Stockfish playing at maximum strength.

9.1 Example game

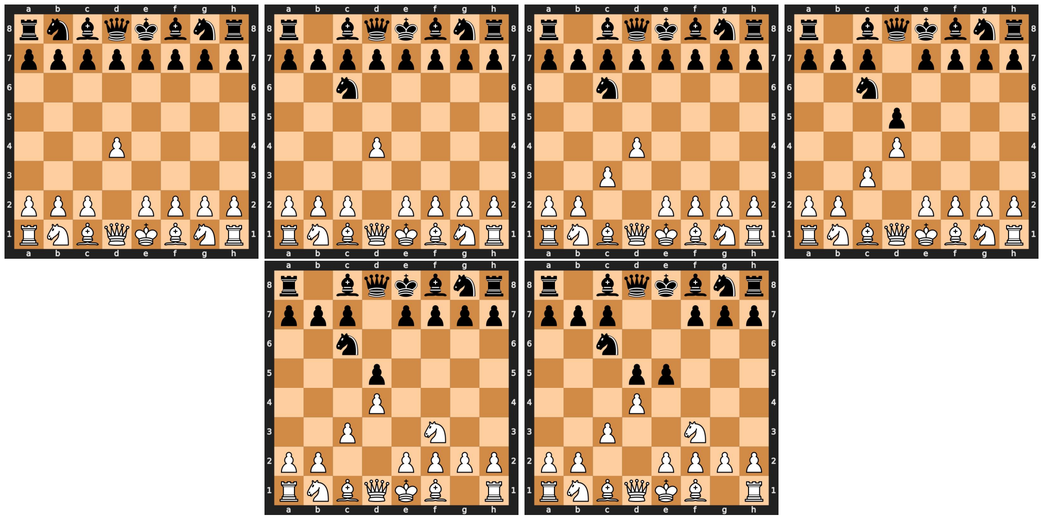


Figure 6: Series of board states showing an example game between human player (white) and AI (black) using python-chess API

10 Discussion

Overall, our model is performing fairly well. It has some strong points and some limitations. We have also identified many improvements which could be made to improve the AI, some of which are directly related to deep learning, and some which are not.

One of the main things that we learned from our results is that the implementation of the search engine surrounding the evaluation function network is highly important to the performance of a chess AI. This means that to develop a fast AI that performs at a very high level, we need to spend considerable time optimizing search, and ensuring that our model can perform inference in a very short amount of time.

10.1 Memoryless model

Being a straightforwards CNN, our model has no ability to gain insights from sequences of board states. While this is not generally a problem, there exist a few edge cases where it is important to consider sequence dependency. These are the threefold repetition and mate-in-50 rules. These rules allow a game of chess to end in a draw under certain conditions, such as the same board position being reached three times in a game or a checkmate being unable to be reached in under 50 moves with one side having only their king remaining. While these conditions are rare, it is important to address them. We propose having a hard coded rule which modifies the evaluation function when these conditions are met.

10.2 Prioritizing important positions and parts of the board

When playing chess, human players are able to quickly identify the most critical parts of a chess board. An expert would spend little time thinking about board features and moves which are unlikely to produce advantageous results. This prioritization is not captured by our model. It would be interesting to explore the performance of a model which used **attention**, for instance, a transformer-based model, in this application.

10.3 Training Time

Another result that was not expected was the amount of time required to adequately train our model. Given the complexity of our model, which was similar to or less than that in other engines we researched, it took many hours (up to eight hours, depending on hyperparameters) to train our model each time we modified it, when training on local hardware. (Nvidia GTX 1070) If a similar project were to be completed with strict time constraints, we would recommend considering using a cloud compute service to train the model.

11 Ethical Considerations

While our project is based on a widely-enjoyed game, it is imperative to consider potential ethical implications:

- Fairness & Bias: The development of a chess AI should be done with the goal of creating a fair and level playing field. This means that our model should not be programmed to cheat or to give an unfair advantage to one player over another. Our model should be trained in a way that is free from any form of bias, such as players' gender, race, ethnicity, or other factors. This requires careful selection of data sources and rigorous testing to ensure that the AI is not biased in any way.
- Privacy & Intellectual Property: The use of data to train the AI must be done in a responsible and ethical manner. This means that personal data should not be used without informed consent and that the data should be anonymized whenever possible to protect the privacy of individuals, which is the case for Lichess datasets. Moreover, potential copyright or licensing issues must be considered when training our model from other third-party sources.
- Transparency & Responsible Use: The developers of the AI (us, the authors of this report) must be transparent about how it works and how it makes its decisions. This can help to build trust among the chess community and ensure that the AI is used in a responsible and ethical manner. Furthermore, steps must be taken to ensure that this model is not used illegally in competitive environments, both amateur and professional. It is simply designed to be a tool for entertainment and is an artifact of our learning and knowledge from the course APS360.
- Impact on the Game & Improvement: The creation of an AI that is capable of playing chess at a high level could fundamentally change the nature of the game. It is therefore important to take inputs and feedback from stakeholders in the chess community. In conclusion, while all efforts were aimed at improving the level of Atomic Chess played by our model, we must acknowledge that our training and model design still has a long way to go and

there is much room for improvement in terms of performance, speed and implementing a Graphic User Interface.

References

Murray Campbell, A.Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial Intelligence*, 134 (1-2):57–83, 2002.

Johannes Czech, Moritz Willig, Alena Beyer, Kristian Kersting, and Johannes Fürnkranz. Learning to play the chess variant crazyhouse above world champion level with deep neural networks and human data. *Frontiers in Artificial Intelligence*, 3:24, 2020.

Andrew Healey. Building my own chess engine, 2020. URL `https://healeycodes.com/building-my-own-chess-engine`.

Tom Kerrigan. Tom kerrigan's simple chess program (tscp). URL `http://www.tckerrigan.com/Chess/TSCP/`.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

12 Appendix

[[0. 1. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 1. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 1. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 1. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 1. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 1. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 1. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 1. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 1. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 1. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	
[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	
[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	
[[1. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 1.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 1.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	
[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]	
[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]	
[[0. 0. 0. 0. 0. 0. 0. 0.]	[[0. 0. 0. 0. 0. 0. 0. 0.]		
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]		
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]		
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]		
[1. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 1.]		
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]		
[0. 0. 0. 0. 0. 0. 0. 0.]	[0. 0. 0. 0. 0. 0. 0. 0.]		
[0. 0. 0. 0. 0. 0. 0. 0.]]	[0. 0. 0. 0. 0. 0. 0. 0.]]		

Table 3: Sample input. The left column represents the current player's pieces. The middle column represents the opposite player's pieces. The last column represents the kingside and queenside castling rights of the current and opposite player, respectively. For example, the top left array displays the current player's pawns