Final year project

Chexplanations: A Chess Explanation System

Chaos

August 4, 2022

Abstract

The purpose of this project is to create an application that is able to generate a human-accessible explanation as to why certain chess moves and positions are advantageous. Currently, the only service that offers this is DecodeChess - A deep learning approach to chess explanation. This project will explore an alternative and less computation-intensive approach based on pattern recognition rulesets inspired by chess literature that has been used for coaching players. Additionally, other aspects of chess pedagogy will be explored with the aim of creating a chess training mobile app that uses the human-accessible explanations generated to give further insight to the user. The explanations generated by the project received overwhelmingly positive feedback during the evaluation stage.

Acknowledgements

I would like to thank my supervisor Viktor Schlegel for guidance with the project for which I am grateful for. The ideas and suggestions that were discussed during the weekly meetups helped shape the outcome of the project significantly.

Note to reader

While is is not required, it is highly recommended that the reader is familiar with the basics of how Chess pieces move. Most of the concepts are demonstrated by example, and knowing the movements of the pieces will help when following the examples. A quick refresher can be found on `https://www.ichess.net/blog/chess-pieces-moves/` (Full credit goes to iChess.net for this article.

Contents

1 Introduction

1.1 Background

Chess, also known as the 'royal game', is a two-player abstract strategy game with no hidden information [1]. Once popular amongst the nobility, modern chess nowadays boasts a much larger player base with over 800 million players [2]. It is also the best-selling board game of all time, with over 3 million copies being sold per year in the US alone [3] . The premise of the game is simple. There are 64 squares with 16 black pieces and 16 white pieces.

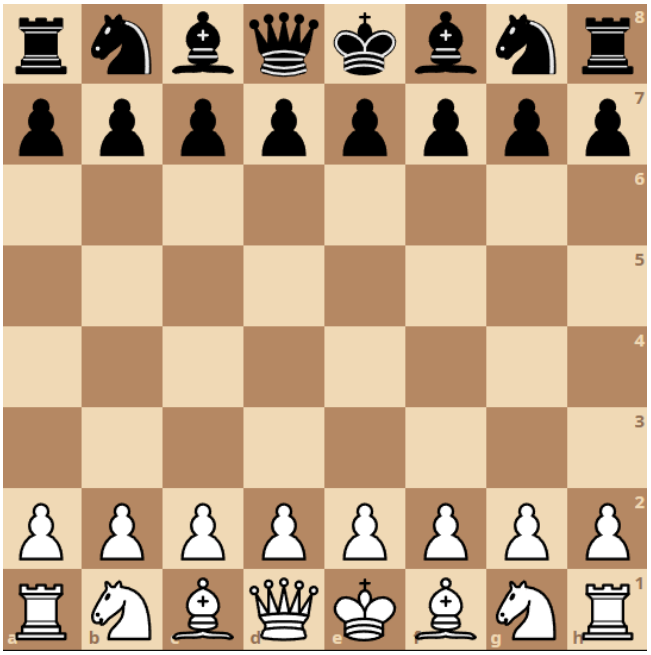

Figure 1: The starting position of a chess game.

The pieces all move in unique ways, for example, the Bishop moves diagonally while the Rook moves horizontally. The Queen is the most powerful piece as it combines the movements of a Rook and Bishop. The aim of the game is simple, you must 'checkmate' the opposing King. That is, you must attack the opposing King in a way such that whatever square it can move to is under the control of one of your pieces.

Despite its simple enough ruleset, the extremely large branching factor means that even with today's best computers, Chess remains unsolved. With a conservative lower bound on the number of possible games being 10^{120} (The Shannon Number), there are more possible games of chess than there are atoms in the observable universe [4]. Because of these properties, not only is Chess one of the most popular board games, but it is also the subject of large amounts of mathematical and computer science research. For a long time, it was unthinkable for a machine to beat a human grandmaster but the fast-paced development of the computer chess world meant that this task was finally achieved in 1996-1997. This was when IBM's Deep Blue defeated Garry Kasparov, the then world champion, in a shocking 3.5-2.5 result [5]. It was this event that marked a changing point in the history of Chess as it was the start of the computer dominance in Chess. Nowadays computers have far exceeded humans in the royal game, to the point where they can start with a serious handicap and still defeat a grandmaster of the game.

1.2 Motivation

Why should we create an application that can explain engine moves? Despite the computer dominance of the game, Chess remains one of the most popular games. Its 'easy to learn, hard to master' nature is what continues to draw millions to the game. With such an impressive player base, it's no surprise that the world of competitive chess has also been growing at a considerable pace with FIDE (Fédération Internationale des Échecs, the governing body) having over 170000 officially rated players in its database. This is just one of the chess federations, there are plenty of national federations too. Once combined, it's easy to see why Chess is considered to have one of the most thriving competitive scenes.

Most chess federation players are rated by what's known as an 'ELO score', a method for calculating the relative skill levels of players in zero-sum games [6]. A difference of 200 points indicates that the player with the higher rating has a 76% chance to win. This means that someone rating 1500 should win 3 out of every 4 games against someone who is rated 1300, with some statistical variance [6]. This score is usually used to refer to the 'strength' of the player.

It is the aim of many players to increase their ELO score or relative strength and they do this via many methods such as books, practice, coaches, and game analysis. Analysing a game with an engine is a well-known method to improve your strength [7]. However, the limitation of such a method is that while the engine can show you where a mistake is, it does not teach you why that is a mistake and the explanation is limited to "White/Black is better" or "This is a blunder" - but the number one question that crops up in the mind of a beginner when faced with such vague comments is "Why?". This project aims to bridge this gap. I aim to improve upon this aspect of chess training to make it more friendly for beginner/intermediate players by giving explanations as to why a certain move is good or what a certain piece is doing so they too can use the power of engine analysis without being hindered by lack of experience or knowledge.

1.3 Project Goals

The aim of this project will be to create an application that is able to explain why a chess move is good. There are very few apps that do this, with the only current one being DecodeChess [30]. While effective, DecodeChess requires not only an active connection but also a lot of computational power due to its deep learning approach. This project will try to create good explanations in a less expensive way. The overarching goals of this project will be to:

- Create a piece of software that will be able to comment on what makes a move that was played good.
- Create an app that incorporates this software alongside popular training methods used by competitive chess players.

These goals are vague, but will be made more specific after the theory behind chess moves is researched and common training methods have been explored.

1.4 Report Structure

- **Chapter 2** will begin by researching what makes a move good by looking at traditional chess literature that is used by humans to improve chess in order to draw inspiration for

rulesets. Then other forms of chess learning will be researched so we can see what the most effective features to include in our training tool are.

- After this initial research has been done, the outline and the objectives for the application will be made more precise in **chapter 3** so we know exactly what to include and what the final application should contain. The technologies used to implement these features will also be briefly discussed.
- In **chapter 4**, the details of the implementation will be discussed including any major decisions that were made along the way.
- After this we will focus on evaluation in **chapter 5**, this is crucial to this project as the project focuses on creating a tool to help chess improvement. In order to see if we have been successful, the explanations it generates must be deemed useful by participants in the evaluation process. This will conclude the main crux of what was achieved by the project.
- The following sections will go over any additional features that I did not get the time to implement fully. I will then end with my reflection on how this project went, how effective I think it is and what I would have done better or improved.

2 Theory & Related Work

2.1 Positional Understanding

As our project revolves around generating human-understandable explanations of chess moves, we need to study chess understanding from the perspective of a human player. Unlike a computer, a human cannot brute force every combination 15-20 moves ahead from any given point to calculate a move. Instead, humans use a mixture of calculation and positional understanding [25]. One could argue that most computers also do this (with the exception of deep learning models), as most heuristic evaluation functions used by chess computers to determine if a position is good or not are based on human positional understanding. However, for the purposes of our project, we will focus more on how humans understand chess - as a beginner/intermediate chess player is more likely to understand what factors make a move good (such as "This move threatens a strong attack") as opposed to "This move is good because it gives a +0.63 advantage" [25]. We will try to find themes and ideas that humans usually use in order to create general rules that we can implement into our system. So with this objective in mind, we will explore classical chess literature that has been used for many years to train players.

2.1.1 Logical Chess Move by Move

The first book I looked at was Logical Chess Move by Move by Irving Chernev[32]. This book was recommended by several intermediate and advanced players on various online websites[9, 10]. The idea of this book is straightforward. It aims to improve a reader's chess ability by example. To do this it goes through 33 master games in full detail - that is, every move explained with the ideas behind certain positions and why some move was chosen over another strong candidate move. Using these games, we can try to generalise common patterns we see as rules to implement in our system.

2.1.2 Imbalances in Chess

"How to Reassess Your Chess" [31] is another wildly popular book and the second one I looked at. This book was also highly recommended as an introduction to more of the positional aspects of Chess. Most beginner players just consider the quality of a position based on the material advantage and using this book we can learn that there are more factors to keep in mind other than that. The book goes into depth about 7 'imbalances' that chess players must keep in mind when evaluating positions. An **imbalance** is a positional difference, an example of such is 'space'. If player A has more space (if they control more squares) than player B, then player A has a space advantage. This is beneficial as it enlarges your influence over the board and provides options that can be exploited tactically and strategically [11]. Here are some examples.

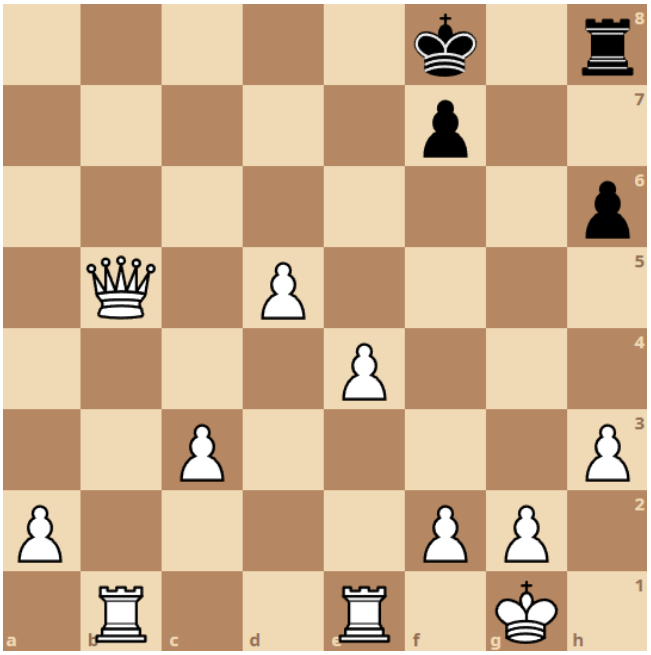

Figure 2: The imbalance here is material. One player has a greater amount of material which them the advantage.

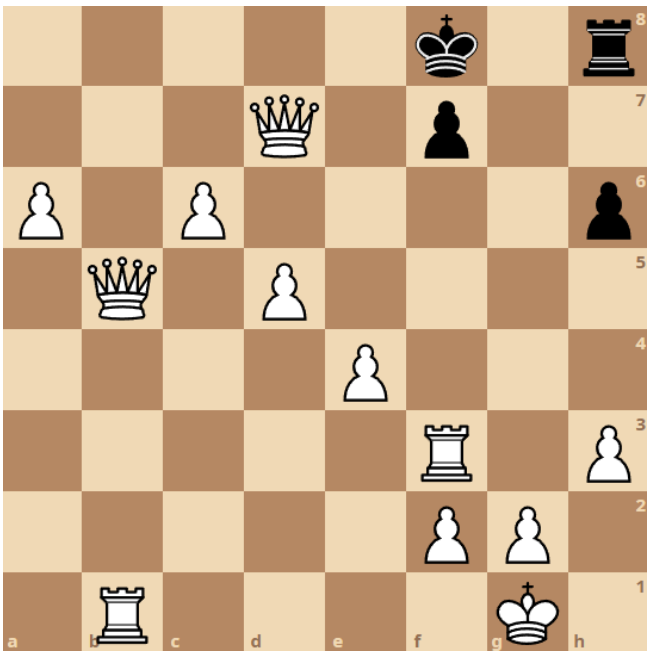

Figure 3: The imbalance here is space (Taken to the extreme). White has such massive influence over the board that blacks options are limited.

The imbalances are as follows:

- Material Advantage, the most well known
- Minor Piece Superiority, Bishops are good in open positions, Knights are good in closed positions
- Pawn Structure
- Space
- Control of centre, open files, etc
- Development
- Initiative

We will implement a few of these imbalances at first with the eventual goal of implementing all of these 'imbalances'.

2.1.3 Initial Ruleset

Our explanation system will take a rule-based approach. From the two books, we have a wealth of rulesets to implement. An example of a rule would be

"If two high value pieces are attacked at the same time" -> "Explain it is a fork".

It will be impossible to implement them all within the time frame of our development. We will first create an initial ruleset that we can implement in order to generate a basic explanation and then we will rely on user feedback to build up the explanation. The initial ruleset we will be implementing is: Material advantage Central Control Basic Tactics - Fork Basic Tactics - Pin The above ruleset was decided based on two reasons. The first reason is that it acts as a

'minimum viable ruleset'. This is because if all 5 of those points are implemented, it would give enough of an explanation to help lots of beginner players. This is because most beginner games are lost because of missed tactics rather than deep positional understanding [12]. The second reason is to not overwhelm the development process by implementing too many rules, the ideal objective in mind is to create a chess training app that uses this engine to enhance the training process by giving human explanations. If too much time is spent on the engine, there will not be enough time for an app. The basic development plan would be to build a minimum viable explanation generation engine, build the rest of the app to a usable state and only then go back and add more rules.

2.2 Studying Chess

Now that we have more of an idea of the deeper aspects of chess beyond piece movement, we will explore how these aspects are trained and studied by competitive players. Investigating these training methods and their efficiency will give us a good indication of what to include in our app.

2.2.1 Useful Definitions

FEN (Forsyth–Edwards Notation): A chessboard is a complex structure. FEN allows us to easily represent any board state with a single string. Below is an example of this notation and how to interpret it.

rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2

The format of this notation is: '"Row1/Row2/Row3/Row4/Row5/Row6/Row7/Row8' 'Who's turn' 'Castling' 'En Passant' 'Halfmove Clock' 'Full Turn Clock'

During the later stages of our report, the first component of this notation will be the main focus - board configuration. The piece codes are: K = King, Q = Queen, R = Rook, B = Bishop, N = Knight, P = Pawn. In the notation, a capital letter denotes a white piece and a lowercase letter denotes a black piece. A number indicates there are that many blank squares. In the above example, on Row 6, the notation states '5N2'. This means there are 5 blank squares from the left, a white knight, and then 2 more blank squares on that row. Using this notation, positions can be described easily.

Evaluation/Eval: This is the score that Stockfish assigns to a position. This score is from white's perspective so if it is positive, that means it's white who has the advantage. If it is 0, the position is drawn. This score becomes better the higher the depth is set. This is because Stockfish works using a minimax tree with alpha-beta pruning to search for potential moves and return the evaluation.

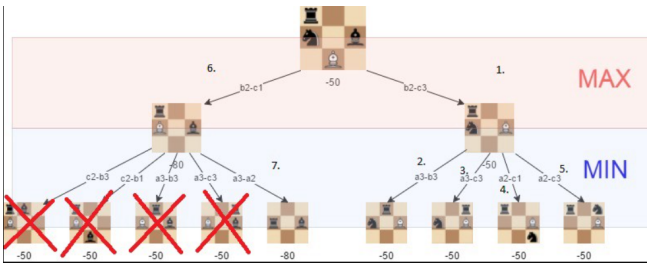

Figure 4: An example of a minimax tree, Photo credits: FreeCodeCamp.

The depth is how many moves ahead in the game tree it searches, the higher it is the stronger the engine. However, it also comes at a computational cost. The depth should be set where the moves returned are very strong but don't take too long to calculate.

Tactics: A tactic is a short sequence of moves that limits the opponent's options and results in a tangible gain [19] such as material advantage or a really important square. The following diagram shows an example of a common tactical pattern known as a '**fork**'.

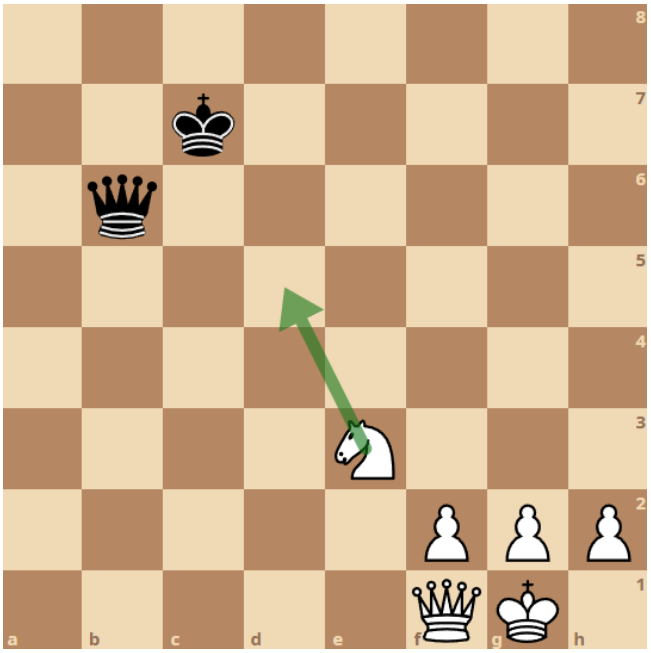

Figure 5: An example of a fork.

A **fork** is when a piece of lesser value attacks two higher value pieces at the same time in a way that the opponent is forced to make a choice to save only one of them. In the diagram above the white player can move their knight from e3 to d5 and attack the black player's king and queen simultaneously. The black king is in check, therefore forced to move the king, meaning the white player can take the black queen the next turn. Even if the white knight is lost in the process (If the black king moved to b7 to escape the check, after the white knight captures the black queen, the black king can capture the white knight), this trade is very advantageous to the white player as a queen is worth much more than a knight.

Strategy: Strategic play and positional play will be used interchangeably in this report. Unlike tactical play, which seeks to gain an immediate advantage through a short sequence of moves, strategic play seeks to build long-term advantages in the position. This is the type of move

beginners will struggle most on, as it requires a good understanding of what makes a chess position good. I will illustrate this with an example. In the previous section, 7 imbalances were described by Jeremy Silman in his book, Reassess your Chess. One of these imbalances was how much space you control and how well you control open lines or files (an open line or file is a row or column of a chessboard with no pieces). With this in mind, assume we have the following position.

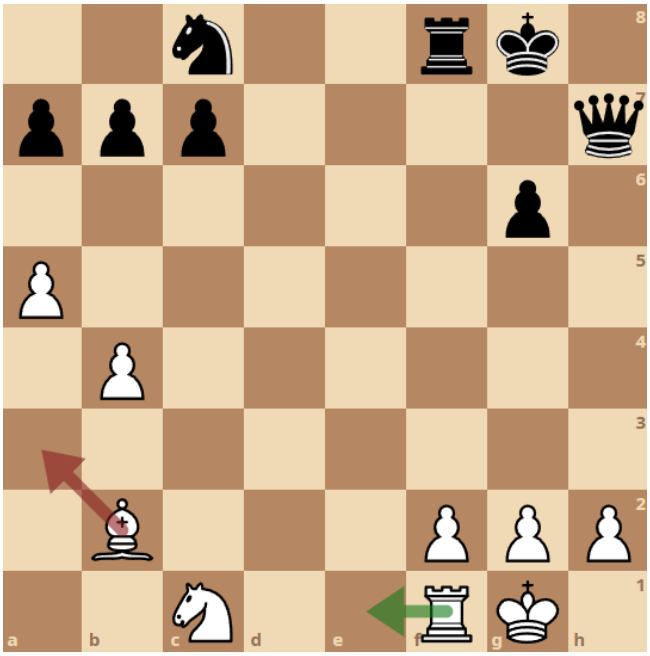

Figure 6: Two different moves with very different strategic consequences.

First, let's consider moving the white bishop from b2 to a3. This is a poor strategic move, the white bishop on b2 is putting pressure on the entire diagonal a1-h8, any black piece that moves into this diagonal must watch out for the white bishop. It also pressures two central squares. By moving this bishop to a3, it loses these advantages, and instead, it does nothing. On the other hand, if we move the white rook on f1 to e1, it goes from doing nothing on f1 to controlling the entire open line e1-e8 and even puts pressure on the centre. This is an example of a good strategic move. As you can see, it does not immediately gain any tangible benefit like winning a piece, it instead improves the position which can lead to more tactical opportunities in the future.

Opening: This is the beginning of the game (for highly advanced players this may last a lot longer). Success in this stage of the game is determined by a sound understanding of opening principles and knowledge of opening theory[27].

Middlegame: The main bulk of the game happens during the middle, plenty of pieces are on the board, and success in this stage of the game is determined by positional understanding and strong tactical vision[27].

Endgame: When most pieces are out of play and only a few remain. Success in this stage of the game is usually determined by theoretical endgame knowledge (such as knowledge of what positions lead to a draw or win) and good tactical vision[27].

2.2.2 Training Platforms

According to iChess.net, the 3 most popular training websites for chess are Chess.com, Lichess.org, and Chess24.com[13]. These are the three largest and most well-known chess servers to date on the internet. Over 77 million people use Chess.com alone to play and improve their chess skills [14]. These platforms offer a huge variety of different training methods which is why they are so popular, we will look at what these different methods are in order to get a better idea of what we might want to include in a training app.

Live games: Like all other activities, the best way to improve is to practice what you have learned in a live game that offers live feedback (For example; If your new opening knowledge or tactical knowledge starts winning more games, that is positive feedback encouraging you to continue). You can also watch high-level games in real-time too, some people claim that watching other people play well improves their own play, especially in the case of streamers. These sites are often used by streamers who play chess and occasionally explain their thought process, giving valuable insight to players[15].

Engine Analysis: All three websites provide engine analysis. This means players will be able to look over any position, or their live games using an advanced chess engine that evaluates the board and points out mistakes for instant correction and feedback. This gives players a clear chance to look at what their weaknesses are during live games and where a transfer is not being seen. To illustrate that last point in more detail, a transfer is where you practice something in one context and apply it in a different context or a bigger context [16]. To give an example of this, a player might be great at solving tactical puzzles individually but might be missing these same tactical ideas in an actual game. Therefore, this gives that player something to work on. It can also tell the player what stage of the game they are most weak on. For example, if the player finds that an engine is telling them that they have a disadvantage near the start of their game for a majority of their games, this indicates that the player needs to work on their openings. Alternatively, if the player loses a lot of their games, but the engine says that the player tends to get a winning position in those games then that player needs to work on converting an advantage (or winning position) into a win. These are just some of the many examples of why engine analysis is an extremely powerful tool in modern chess improvement. However, there is some room for improvement. When an engine suggests the correction for a mistake or other good moves, it may not be immediately obvious why that move is good for beginner human players. As stated in the project proposal, our aim will be to use the positional concepts explored in the previous section to improve engine analysis by providing these explanations.

Puzzles: Training with puzzles is one of the most effective ways for a player to improve their tactical vision in chess. Tactical vision is a chess player's ability to not only spot and execute tactical ideas but also defend against them too. A key aspect to improving this is to improve pattern recognition [17]. Although there are many, many different configurations of a chessboard - tactical ideas can have similarities in their structure. Consider the following diagram where the white queen is moved to d2.

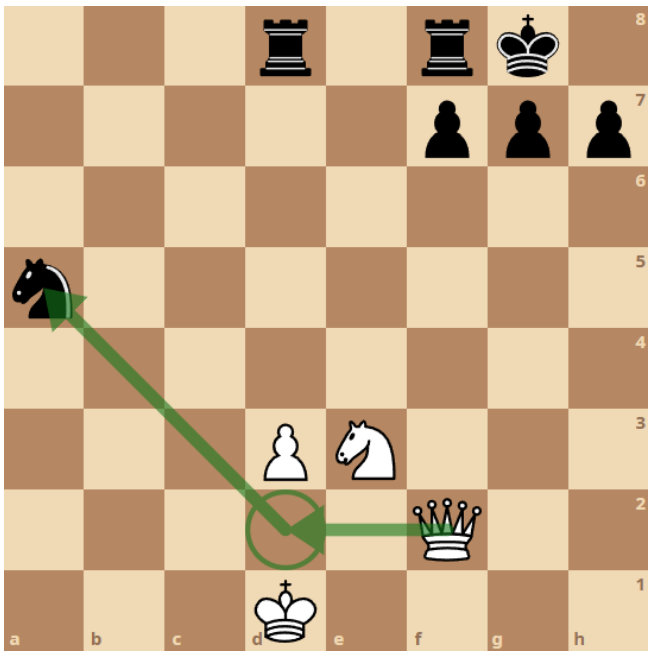

Figure 7: Tactical Vision.

A beginner's reasoning behind this move would be along the lines of "That knight is undefended, I'll attack them with the queen and force them to do something about it". An intermediate or advanced player would think this but in addition to this, they will also think "This queen is now dangerously lined up in front of my king. The only thing stopping the opponent's rook on d8 from attacking my queen is that sole pawn on d3. I must be careful to not let that pawn fall and move my queen to a safer square if possible." The best way to build up the level of tactical vision/awareness is to play lots of games and solve lots of puzzles. This will help you build up a bank of 'patterns' that often lead to tactical blows (patterns such as pieces being lined up dangerously, like the above example. Another example would be the fork mentioned earlier). The importance of drilling puzzles during chess training has been so greatly realised that entire websites and servers dedicated solely to solving puzzles have been created, such as ChessTempo. The three leading chess servers have also been innovative with puzzles and offer a variety of ways you can solve puzzles to keep it entertaining and encourage more people to do them. For example, Lichess.org now offers modes where you can race against other users to see who solves puzzles faster and modes where you solve as many as you can in a given time frame (Puzzle Storm). Websites like ChessTempo which specialize in puzzles give an even greater variety of features, such as specific puzzles for specific parts of the game. It is safe to say that puzzles have played a crucial part in how a majority of competitive chess players train online.

Tutorials and lectures: All three sites have interactive tutorials and/or a video library filled with instructive content from top chess players. The sites also offer one-to-one coaching with top players for a price. Below is an example of the Lichess beginner learning page.

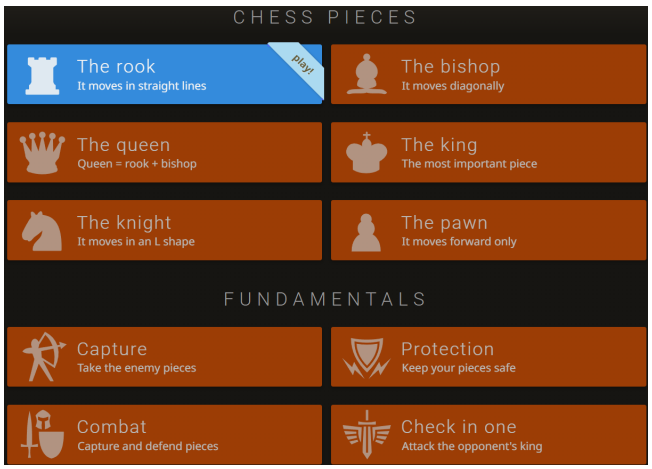

Figure 8: Lichess Training Page.

This page is designed to take someone with no knowledge of the pieces to a beginner level of strength provided they follow all the example positions.

Opening Explorer: As mentioned earlier, success in the opening state depends not only on the understanding of opening principles but also on knowledge of opening theory. Opening explorers can help you learn what opening moves are good or not and how to respond. Let's say you are used to playing a certain opening sequence but the opening explorer says that for one of the moves in your sequence, the database says that move loses 85% of the time. That means this is not a good move to be playing and needs to be changed. The player can then look at the opening explorer to see what the best move in that position is and adopt that instead. Doing this over and over, while also referring to opening theory books will allow the player to build up an 'opening repertoire'. This is analogous to an 'opening book' for chess engines. Most engines have an opening book, this is a table of known good moves that the engine refers to during the opening stage because there are far too many moves in the opening to search through and evaluate. Likewise, most human players don't calculate new moves in the opening stage of every game, instead, they have studied and learned sequences of opening moves and how to respond to the opponent's best replies in order to mentally recall them quickly. This way, the only time a player would need to start working out new moves is when the opponent plays something the player has not studied before or when the player exhausts their opening repertoire (most intermediate players only study 5-10 moves deep, so after that, they can no longer rely on their memorised opening sequences).

All three websites provide a wide array of different training methods for different stages of the game as explained above, so they will provide a useful reference point for what we should include in our training app. Lichess had an exceptional study section where users can explore whatever features they want, something which Chess.com. This feature will also be included in the app.

2.2.3 Spaced Repetition

In the last section, memory was mentioned in regards to opening sequences and moves. Chess players can get by without memory as a beginner or intermediate player, evening in the opening stages by relying on understanding the principles rather than memorising sequences. However, as a player advances further into the study of chess, there comes a point where you must study opening theory. Because this becomes a necessity at some point, lots of memory-based training software has become available recently. The technique that underpins most of these

applications is known as ‘Spaced Repetition’.

Spaced repetition is a method where the subject is asked to remember a certain fact/object with the time intervals increasing each time the fact is presented or said [20]. This technique has been shown to be very effective for memory recall [21]. A brief summary of the science behind this method is that it takes advantage of the ‘spacing effect’ and suggests that items are processed more deeply if we try to retrieve them just as we are about to forget them (Retrieval effort hypothesis) [22].

Several chess training apps have used this concept. The first is Chess Position Trainer. This is a very effective and simple app. The idea of this app is you enter your opening repertoire, that is, all your opening sequences and your replies to your opponent’s viable responses. You can then use the training feature which will then play your opponent’s viable responses and your task is to play your reply to that response in order to further strengthen your memory of it [23]. This process will be illustrated with a simple example. In the following position, you play white and start with the move pawn to e4. If black responds with pawn to d5, you will capture that pawn. If black responds with pawn to e5, you will move your knight on g1 to f3.

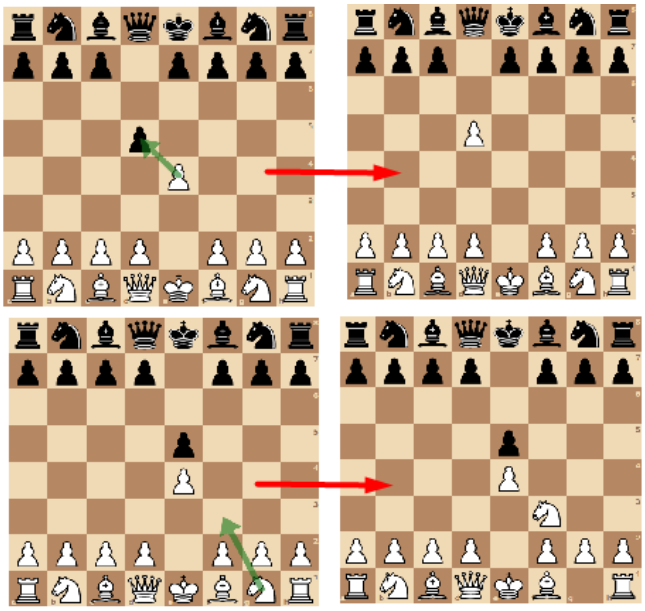

Figure 9: Sample opening sequence.

To use this app, you would enter both of these sequences and then use the training option. Now when you play pawn to e4, it will pick one of the two responses you entered randomly and your task is to play the correct reply. If you play it correctly, it will increase the time before you are next tested on that specific sequence in accordance with the spaced repetition method. This is a very simplified example, advanced players have 100s to 1000s of sequences that may be 10-20 moves deep and software like this really helps organise it and practice it in a systematic way.

Another noteworthy service is Chessable, an online spaced repetition based chess trainer that has exploded in popularity recently and has been endorsed by the world champion Magnus Carlsen himself [24]. This service allows popular chess masters and grandmasters (and other regular users) to submit ‘courses’. An example of a course would a ‘Ruy Lopez Short and Sweet Opening Course’.

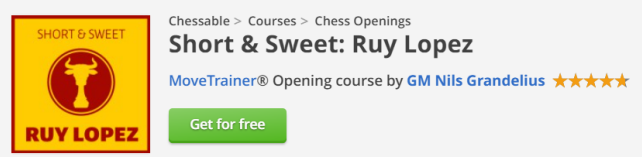

Figure 10: Chessable Course on Ruy Lopez by Grandmaster Nils Grandelius.

This course will give a player immediate access to 20 high-quality opening lines (opening lines and opening sequences are used interchangeably) with a full explanation of why the moves made in the lines are played and allows the players to drill these lines into memory using a spaced repetition system. This gives any player the chance to quickly and effectively add grandmaster endorsed opening lines to their current opening repertoire. As a result, it has changed the way openings are trained in chess considerably. It is now a mainstream tool that many popular grandmasters contribute to and millions use.

With these two examples, we can see the effectiveness of spaced repetition and this makes it an ideal candidate for a training feature. Our app should use spaced implementation for puzzles but should not implement any functionality that helps train openings. This is because Chessable is already at the top of opening study tools available and there is not much our app can improve on.

2.3 Stockfish

The aim of this project is to create an application that explains engine moves. This project will use Stockfish as the engine behind move generation. Stockfish consistently ranks near the top of chess engine rating lists and is an 11-time chess engine championship winner [8]. The reason Stockfish was chosen was because of the fact that it is open source and has an active community.

3 Project Requirements

3.1 Success Criteria

We will now make clear exactly what we need the explanation software and the app to achieve based on our research.

Explanation software:

- Must comment on tactical ideas. At the very **least**, it must comment on the two in the initial ruleset - forks and pins.
- Must comment on positional ideas. At the very **least**, it must comment on the two in the initial ruleset - material and centralisation.
- Must give readable and **viable** explanation. "This move threatens to take the queen" is an example of an viable explanation that humans, especially beginners, can understand. "This move gains positional advantage in 54 moves along the tree" is not comprehensible by humans.

App:

- Must incorporate the explanation engine
- Must be able to fetch computer moves
- Must incorporate tactics with spaced repetition
- Must be able to study positions like Lichess study feature.
- Must be easy to use

3.2 Deliverables

Technically speaking, there are two deliverables. The first, and main, deliverable is the explanation generating engine. This can be used as a stand-alone engine with its features being accessible directly or from an API. This can be used by other chess applications in order to add some explanation to moves being made.

There will also be a mobile app that incorporates the well known chess training methods that have been researched. This not only gives a player various training methods, but also servers as a demonstration of the explanation generation software in use in conjunction with these methods.

3.3 Technologies

Python is a high level interpreted language. I chose to use this for the explanation generation engine because of a few reasons. Firstly, Python has a huge community and lots of different libraries that I can easily use. The main reason I used to was because I was very familiar with it, therefore it was a productive choice to use for me. I will be able to quickly implement and test different components of the explanation generation.

SQLite is a 'small, fast, self-contained, high-reliability, full-featured, SQL database engine' [28]. This is an unconventional choice as there are better and more robust database libraries out there. The reason I needed a database library was because I needed to store puzzles for spaced repetition. This is not really critical data, and the way I will be using it does not require any advanced features. Therefore I chose to use SQLite for its simplicity and for the fact it was a native library in Python.

Stockfish.py is a library makes it possible for python to interact with the Stockfish engine. The documentation is located at https://pypi.org/project/stockfish/. This library will be useful for whenever I want to generate a 'Best Move' in accordance with the engine. This functionality is used for generating computer play and for generating hints for the player. If the player does not know what move to make, they should be able to request Stockfish to make a move and then use the explanation feature to see why that move was made.

While the explanation engine was being done in Python, Python was not very well suited to making a mobile app so I chose to use something different for that purpose. However, whatever I use for the app will need to be able to communicate with the explanation engine somehow. Therefore I chose to use Flask. **Flask** is a micro web framework that is well suited for making APIs. I learnt how to use Flask to create an API for my explanation engine so that whatever I ended up using for the app will be able to just send API requests in order to get the appropriate

response back.

React Native is mobile application framework. I chose to use React Native because React Native apps are faster to build and it is easy to build apps in a modular way with reusable components. The components are built using a mix of JavaScript and the React Native language. This method of building components was something I was not familiar with so I spent 1-2 weeks reading the documentation and creating some simple apps in order to get used to the workflow.

4 Implementation

4.1 Overview

The features implemented in the app are as follows.

- Standard Functions - Hint/Undo/Reset
- Fetch explanations.
- Fetch Incorrect Explanation to Test users.
- Load positions via FEN.
- Creates puzzles from users games.
- Allows training of said puzzles in a way that facilitates spaced repetition.
- Explains Piece Roles in a given position.

The explanation software was built iteratively, it started off with the initial ruleset then expanded its ruleset based on continuous feedback. The explanation software comments on the following factors by the end of the project:

- Material Advantage
- Central Control
- Central Attacks
- Piece Activity
- Tactics - Forks/Pins/Discovered Attacks
- Piece Interactions
- Pawn Structures
- It can explain Piece Roles

Both the app and the explanation software are made up of many components which work together. In the software's case, FEN and move requests are passed through the corresponding components which generate sentences. These sentences are then formed into a paragraph that is returned. The next section contains detailed diagrams to show how these components interact.

4.1.1 Design Diagrams

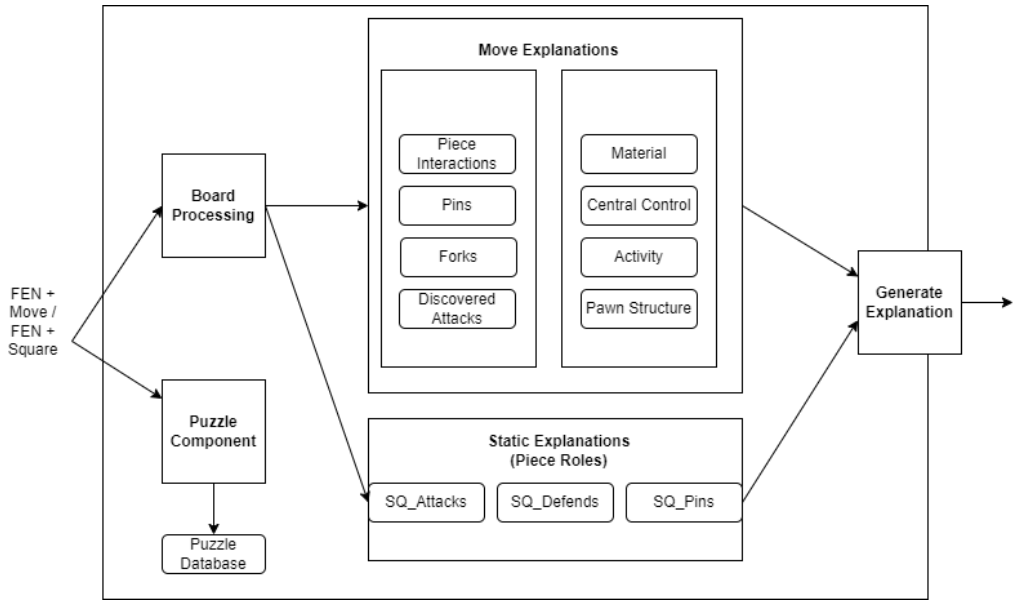

Figure 11: A diagram of the component interaction in the explanation software.

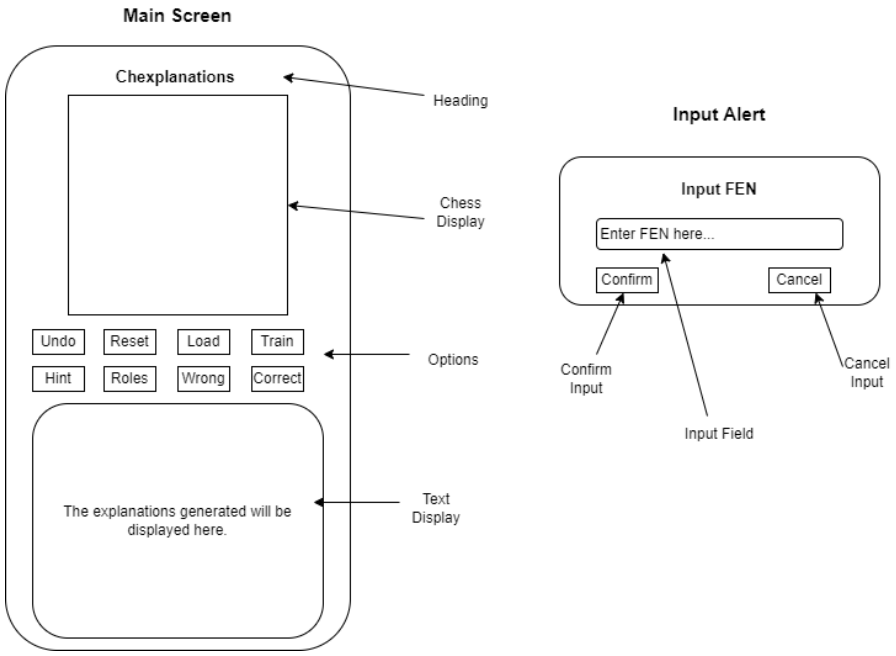

Figure 12: The design plan for the app along with all its intended features.

4.2 Puzzle Component

We will now look into the components in more detail, starting with one of the smaller ones, the puzzle component.

4.2.1 Puzzle Generation

During the earlier research on the different training methods used, puzzles were one of the most used and most effective ways to study chess. Using the board evaluation feature, puzzles

can be generated. The steps to do this are as follows:

```
Set a new board with any arbitrary position
Set player 1 to a Stockfish instance with 18 depth
Set player 2 to a Stockfish instance with 10 depth
Repeat until checkmate:
    Player 1 makes a move
    Player 2 makes a move
    If evaluation changes by more than 2 points
        Record position as a mistake in the database
```

This sequence is repeatable as many times as necessary, providing access to an infinite number of puzzles. However, there was a problem with this approach. It is difficult to quantify the problem with words but after feedback from users and personal testing, the puzzles had an inhuman quality to them. Because the computer generates all the moves, lots of the positions it reaches aren't the style of positions humans reach. A different method had to be used.

A personalised approach was taken. Instead of generating puzzles via two computers playing, The script instead uses the same idea but moves the puzzle detection aspect to when a human user plays a move. While the drawback of this method is that there will be far fewer puzzles for the user to solve, each individual puzzle that does get added to the database will be much more beneficial to the user. This is because correcting your own mistake is often more valuable than solving alien computer-generated positions.

4.2.2 Refutation

The solution to a puzzle is not always obvious. The research covered a variety of training platforms and when a mistake was made, the feedback from these platforms varied. Here is an example:

The idea behind the puzzle: Your queen is attacked, you must defend it. If you play a move that does not defend the queen, the feedback will show the move is incorrect with an arrow indicating the opponent taking the queen. In this case, it is obvious what the mistake was.

But what if for a different case the solution was not simply defending a piece but securing a long-term positional advantage in which you'll only see the consequences 3-4 moves down the line? In this case, the platform informing the user of an "Incorrect Move" with the arrow indicating the next move doesn't really explain much to the user.

This lack of feedback can be improved upon with the following process:

```
Set the position of the puzzle
Take the user's input move
If the move is not the correct solution
    Until the evaluation shows an advantage
        Play best move
```

Use explanation generation to explain why the opponent now has a better position

This process will take the user's move, and if it is wrong it will play the best possible continuation. It then explains why the opponent is better in this perfect continuation which will give insight as to why the first move that led to this conclusion was not optimal.

4.2.3 Puzzle Storage

The script currently generates puzzles but has no persistence. If the script were to be closed and reopened, all data is cleared. To make it persistent, all the generated puzzles will be stored in an SQLite database. SQLite was chosen due to it being a native library in python and very easy to use.

Every puzzle will be stored in the form:
<FEN of the puzzle position> <Solution> <Counter>

The reason why the solution is stored alongside the position is that it reduces the number of requests needed. This script (which is part of the engine) will be accessed by other applications, such as the training app. If just the FEN of the puzzle position was stored, whatever program that needs to access this service will need to send 2 requests. One to fetch the puzzle, and another to calculate the best solution. If the solution is saved to the database from the start, the puzzle and the solution can be fetched simultaneously.

The counter is there to facilitate the use of spaced repetition, when a puzzle is saved for the first time it is set to 0. Every time that puzzle is solved correctly, the counter is incremented to indicate the number of days before that puzzle needs to be solved again. If the puzzle is ever solved incorrectly, this number is reset back to 0.

4.3 Engine Implementation

4.3.1 Board Processing

The explanation generation works by applying certain rules and scoring systems to a position. This is hard to do on a FEN representation of the board such as 'FEN'. Therefore, the FEN will be processed into something that is much more easily workable such as a two-dimensional array. To do this we will use the following algorithm.

Begin Algorithm - Board Processing

1. Split *FEN* and extract first component, this is the board '*layout*'.
2. Split '*layout*' into 8 strings, one for each row. Call this list of strings '*rows*'
3. Set '*board*' = []
4. For *i* in *rows*:
    4.1. Set *currentRow* = []

4.2. For *j* in *i*:
If *j* is a number: *currentRow.append(‘ ‘ x j)*
Else *currentRow.append(j)*

4.3. *board.append(currentRow)*

End

This algorithm will take a FEN string, and represent the layout as a 2d array, it does this by iterating over all the row representations and assigning items to the list accordingly. For example, if row 4 is represented by ‘5Q2’ it will append ‘ ‘ x 5 (5 empty strings) to the list, followed by ‘Q’ and then ‘ ‘ x 2. The resulting currentRow list would be

[‘ ‘, ‘ ‘, ‘ ‘, ‘ ‘, ‘ ‘, Q, ‘ ‘, ‘ ‘]

The software needs a way to translate moves between the notation used by the board (eg. ‘e2e4’) into something that can be used in Python, as ‘e2e4’ is not a valid index of an array. To do this, a dictionary was used. The dictionary contains letter and index pairs (‘a’ :0, ‘b’:1, ... , ‘h’:7). Let’s call this dictionary ‘file2py’ and let’s say we have a certain square on the board, ‘e2’. We can use this in the formula below to obtain an index for a square in our 2d representation:

location = [8-square[1],file2py[square[0]]]

Using our example of ‘e2’: Location = [8 - 2][4] = [6][4]. The board below can be used to verify that this is correct.

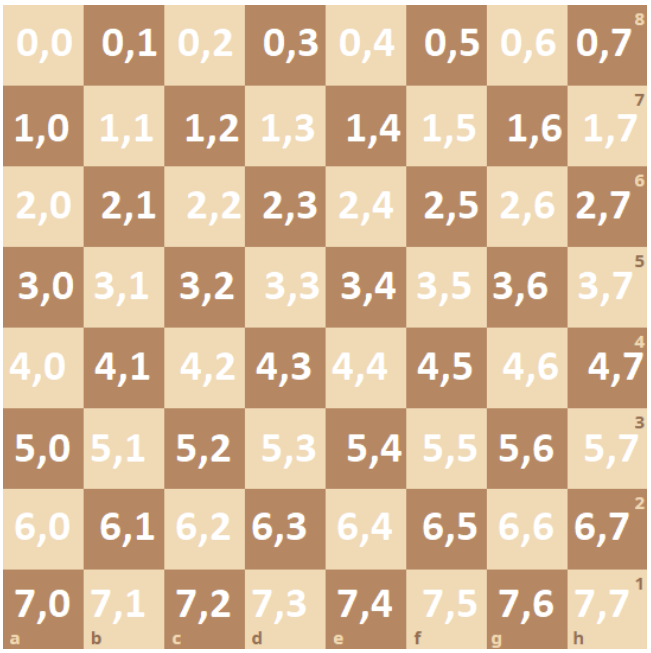

Figure 13: Board squares to 2D index notation.

An equivalent formula can be used to translate from the index notation back into the boards notation by the following formula where py2file is the dictionary file2py with the keys and

values swapped.

Board Notation = py2file[location[1]]+str(8-location[0])

The tactical component will need to be able to access a list of moves in order to look for tactical motifs such as double attacks. Unfortunately, there is no way to derive the rules of Chess from the FEN provided as it is just a form of board notation and nothing more. Therefore, a movement function is needed to extract possible moves from a certain square for easy use later. This function will be work as follows

Begin Algorithm - Legal Move Extraction

1. Take in board (2d array) and square (string, eg 'e4')
2. Use conversion to convert square into index notation and locate it on the board
3. If square is NOT empty:
    3.1. Determine piece on that square
    3.2. Calculate all legal moves in accordance with the piece rules.
4. Else
    4.1. Return empty list as there are no pieces on that square

End

This algorithm simply calculates all the legal moves from a certain square on the board. All the piece movements are hard coded so for example if the piece on the square was determined to be a bishop, then it would loop through all the diagonals until it cannot move any further (either because it would be outside the board or another piece blocks its path). We will use this a lot in our tactical component in order to spot common patterns.

4.3.2 Strategic Component

We will now build the first major component, the strategic component. This component has a mix of scoring-based sub-components and pattern-based components. A pattern-based sub-component scans the board and sees if pieces fall into a certain configuration. A scoring-based sub-component looks at the board as a whole and applies some sort of scoring mechanism to determine if there is some advantage or disadvantage in the position.

Material Advantage
We will begin by implementing the first scoring-based sub-component. To determine the material advantage in a position, we first need to determine what each piece is worth. The traditional piece values will be used for this, these values are:

- King: ∞, since it cannot be taken. This value will be represented as 99.
- Queen: 8. This is the most powerful piece on the board.
- Rook: 5. Usually considered more powerful than a Bishop or a Knight.
- Bishop, Knight: 3.

- Pawn: 1. This piece can promote to a better piece if certain conditions are met.

To score the board, the software will start at a material value of 0 and then loop through every square on the board. If a certain piece of its own colour is found, it will add its value to the material value. If a piece of the opposing colour is found, then it will subtract the value from the material value.

Now the software needs to be able to differentiate between different levels of disadvantage. Being down an entire queen is a lot worse than being down a single pawn, and the explanation should reflect that. To do this, hard coded thresholds are used. These thresholds were tested by showing the explanations to chess players on the chess servers and the chess society. These thresholds were then adjusted based on the feedback received. The final thresholds were 0, 3, and 6. If the material value is above 0, you have a slight material advantage. If it is above 3, you have a strong material advantage. If you are leading by over 6 points this is an overwhelming material advantage. Conversely, if you were down by this many points it would be a slight/strong/overwhelming disadvantage instead. If the material is equal, no comments will be made. This sub-component will be used to inform the player of a material advantage or disadvantage if one exists.

Central Control
Central control is a good positional advantage to have as mentioned in Jeremy Silman's Reassess Your Chess. This is the second scoring-based sub-component. We will use a simple scoring mechanism where the most points will be allocated to the centre of the board. We will also allocate some points (but fewer) to the outer ring that encloses the 4 central squares. In this first version of the component, the feedback indicated that it was not very precise. Having 3 knights in the centre was better than having 2 pawns. Therefore, it was adjusted further to give these final values for each square. The value on the right is for a pawn, the value on the left is for a piece.

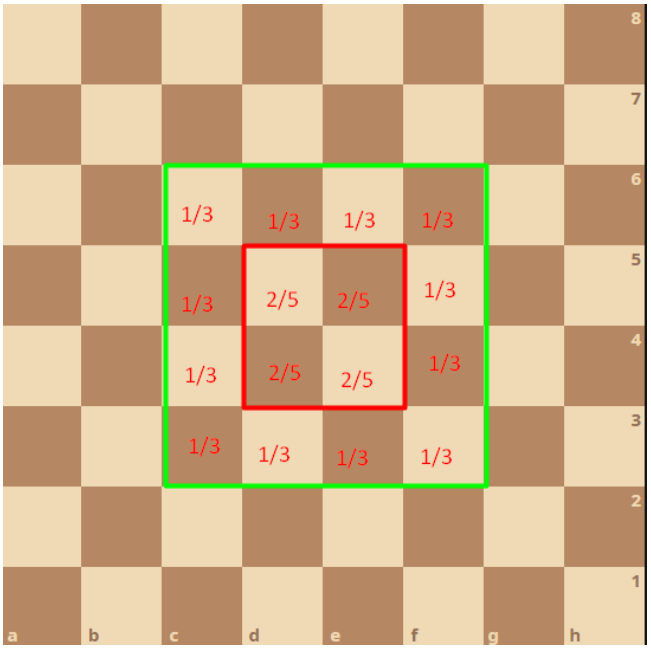

Figure 14: Central values reached after testing.

This second iteration was much more successful but there was an issue with one piece. This

piece was the King. The King counts as a piece so it was included in the calculation for centralisation. If the King is in the centre of the board while there are lots of pieces out, this is incredibly unsafe. This is because the more pieces there are, the more possibility of there being an attack and if the king is in the centre, it is usually dangerous letting it become the potential target of many attackers. However, the king cannot just be simply removed from the calculation because that leads to another flaw. If the game is in the endgame stage with a few pieces on the board, then a game can be decided solely on the fact that you have an active, centralised king or not. This is because, during the endgame stages of the game, the amount of pieces on the board is greatly reduced which lowers the possibility of being attacked.

To solve this, the software uses a special case for the king. It will count the number of opposite coloured pieces on the board (excluding pawns) and then if that number is less than 2, a centralised king will benefit from the position like a regular piece would. On the other hand, if there are more pieces then the score for centralisation will remain the same however a flag will be raised to indicate the king is unsafe. This flag will be used in the explanation generation to give comments about king safety. Now, this sub-component works as intended, it can be used to detect who has better centralisation and flags up king safety issues.

Positional Factors
There are three more positional factors we will implement. All of them will be mentioned in this paragraph as they are individually simple to implement.

Piece Activity - To see if a piece has become more active, we can count the number of legal moves it had before the move and the number of legal moves it had after the move. If it has more after the move, then the piece has become more active.

Pawn structure is a vast topic, our explanation engine will only focus on **doubled pawns** and **passed pawns** for now. A doubled pawn is seen as detrimental as they cannot defend each other whereas a passed pawn is a pawn with no pawn to oppose it which is strong as it will have an easier time promoting.

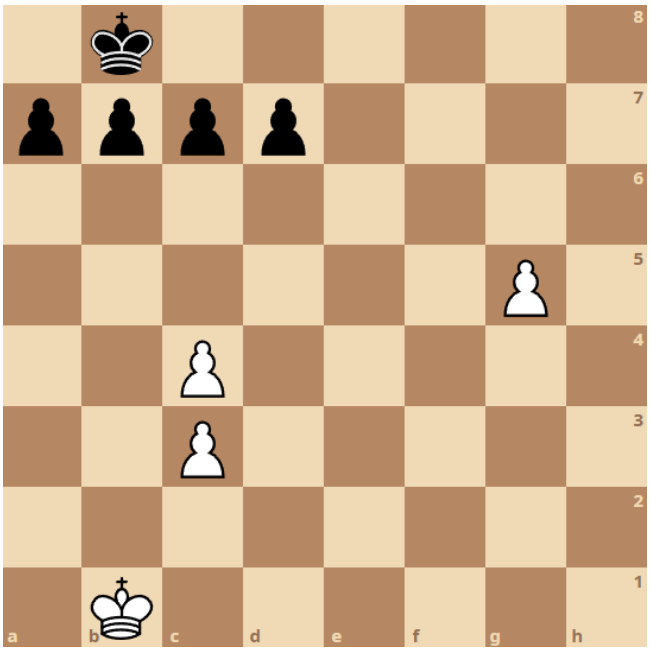

Figure 15: The c4 pawn is doubled and the g5 pawn is a passed pawn.

Doubled pawns - every time a piece is taken, check if it is a pawn. If it is, check the column behind the pawn (one of the indexes will always be a fixed so a simple loop was used) and if a same coloured pawn is found then a doubled pawn has been found. The same technique can be used to find passed pawns. When a pawn has been moved, check the column in front and if no opposing coloured pawn is found, the pawn that just moved is a passed pawn.

4.3.3 Tactical Component

We will now build the second major component, the tactical component. Unlike the strategic component, the sub-components in the tactical component are all pattern-based. These sub-components focus more on the interactions between the pieces than they do on the overall board position.

Attack and Defense

The first interaction we have to deal with is the simplest; attack and defense of a piece. This may seem like something that's too obvious to implement. After all, it should be obvious when you are attacking a piece, right? Yes, but that's not the whole picture. Board awareness is one of the most vital skills that a beginner needs to develop to reduce blunders. A beginner may move a piece immediately to defend another piece but forget that the piece they just moved is threatening something even bigger leading to missed opportunities. Here is an example of that.

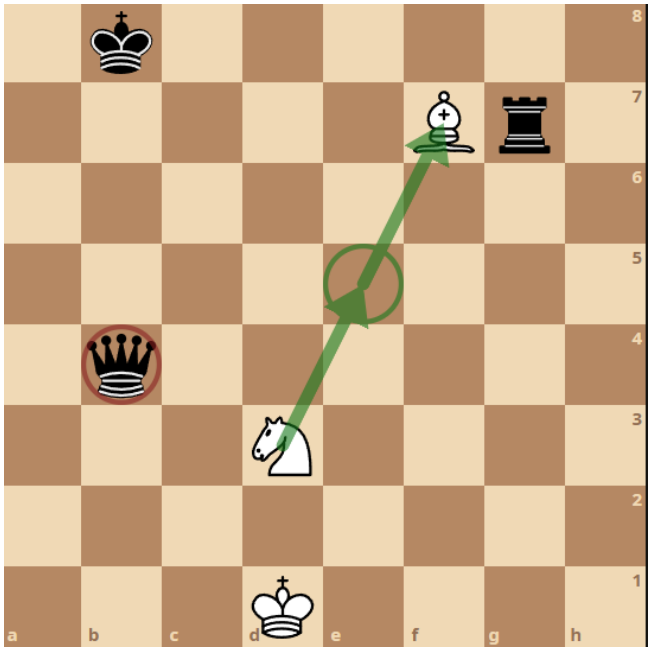

Figure 16: Rushing an attack can lead to missed opportunities.

The white bishop is in danger, so the white knight rushes to its defense. However, instead of this move, the white knight could have captured the black queen - a much better move as it is the most powerful black piece! Another example of a lack of board awareness is rushing to attack an opposing piece and forgetting the defensive role that the piece plays. In the following example, the black knight attacks the white rook but in doing so, it gives up the only defense of the black queen, which can now be taken by the opposing queen (had the knight not moved, if the white queen captured black's queen then the black knight can capture the

white queen for an equal trade).

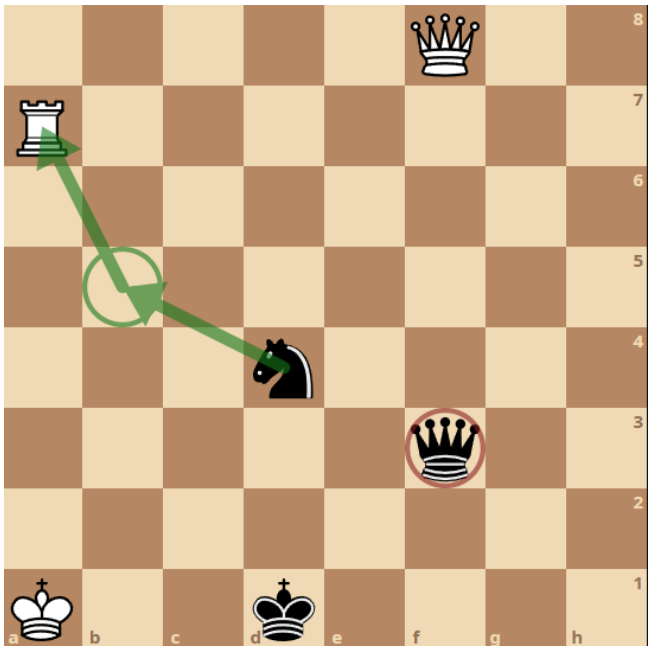

Figure 17: Careless attacks might stop defending key squares.

Training board awareness will help eliminate these types of errors. To do this, on each move the software will take note of what get's attacked, defending and, what pieces you have lost defense of. This may seem like a lot of information, but board awareness is such a crucial skill to develop that this was the first sub-component that was developed. It will take in a FEN and a move string (for example, "e2e4"). It takes the second square of the move string (in the "e2e4" example, this would be "e4") and then calculates all the legal moves from this position. It then loops through this list and if any same coloured piece is found, it is added to the '*defended*' list (except the king, because a king cannot be taken by an opposing piece so it cannot be directly defended). If an opposing colour piece is found, it will be added to the '*attacked*' list. Any central squares will also be detected and added to this list because attacking the centre increases pressure on the centre which is in turn indirectly controlling/influencing the centre slightly.This list can be used in the explanation generation to generate the board awareness sentences of what's being attacked and defended.

To indicate what is no longer being defended requires more work. The function that retrieves the legal moves given a square needs to be adjusted. Instead of extracting the piece from a given square, it can now also take in a piece as a parameter (by default it does not do this) to overwrite the piece extracted from the square passed in. What this means is, now if we pass in any square and a piece, for example, 'N' (a white knight), it will calculate the legal moves as if the knight was there. This allows us to calculate an '*old_defended*' list by using the first square in our move string (in the 'e2e4' example this would be 'e2'). We can then compare between '*old_defended*' and '*defended*' to see what is no longer being defended. With these three pieces of information together, all the required board awareness sentences can be generated.

Double Attacks

A double attack is a natural extension after the attacking sub-component and relies on the same '*attacked*' list the board awareness sub-component uses. This tactical pattern is also commonly referred to as a 'Fork'. This is where one piece attacks two or more higher-value

pieces at the same time. The chess explanation software should be able to state when this case happens so the user is aware two higher value pieces are being attacked (the standard piece values will be used). To achieve this, we will use the '*attacked*' list that was calculated before and if there are two or more pieces in this list then the list is filtered down into pieces with greater value than the piece doing the attacking. If there are still two or more pieces in this filtered list, it will be used to generate a sentence indicating that there is a simultaneous attack with more valuable pieces that the user should pay attention to (the actual sentence generated will be more precise in its wording, but that will be covered later).

Pins

A pin is where a piece is attacked but is unable to move because if it moves, then a more valuable piece behind it gets attacked instead. The following is an example of this type of tactical pattern.

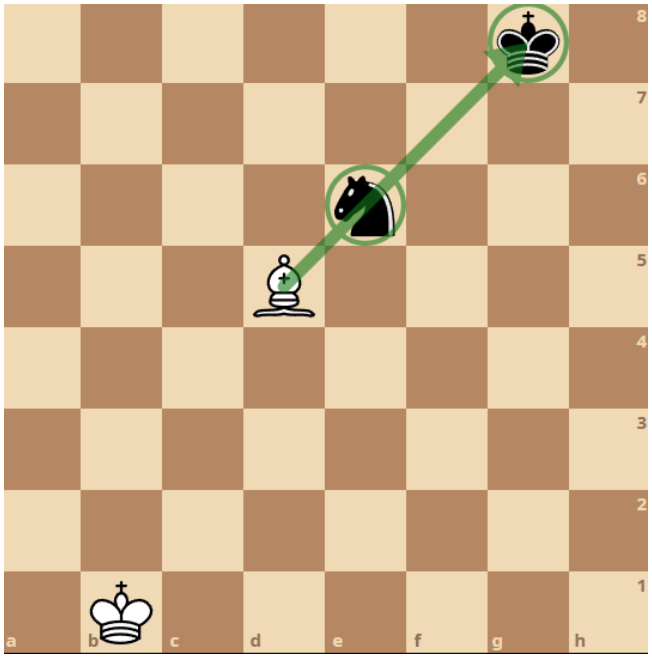

Figure 18: An example of a pin.

In the example above, the white bishop pins the black knight. The black knight cannot move because it would expose the king to danger. This is an example of a strong pin. A absolute pin is where the piece (in this case, the black knight) physically cannot move because it would be an illegal move to do so (as the king will be put in danger). Suppose instead of the black king behind the knight, there was a black queen. This is an example of a relative pin. In a relative pin, the piece can move but it is a bad idea to move it (in this example, moving the black knight would result in the loss of the black queen).

To implement this, a modified movement function needs to be created. Currently, the movement function stops when it reaches the bounds of the chess board OR the path gets blocked by a piece. The modified movement function adjusts this so that it ignores a piece blocking a path and only stops if a second one blocks the path. If this occurs, it appends it to a list. The elements of this list will have the form [*piece1*, *piece2*]. This list is filtered with the following condition:

If *piece1* and *piece2* are both the opposing colour AND value of *piece1* < value of *piece2* then keep the element. Otherwise, discard.

Any remaining elements after the filtering process has occurred gets used in the explanation generation. This method of discovering pins allows for a way to detect another useful tactical pattern, the discovered attack.

Discovered Attacks
A discovered attack is where you can move a piece and another piece starts attacking a more valuable piece of the opponent. This is clearer with an example.

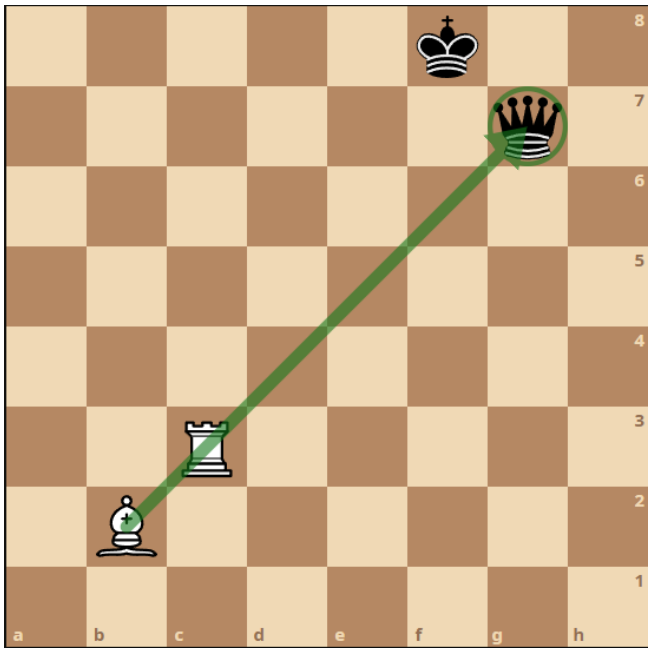

Figure 19: An example of a discovered attack. The white rook is the only thing stopping the white bishop from attacking the queen.

If the white rook moves anywhere, the white bishop will start attacking the black queen. This example is particularly devastating as the white rook can move to c8 to attack the black king, putting it in check and forcing it to move. This then allows the white bishop to freely take the queen next turn. As shown above, this can be a devastating tactic if found.

To implement detection of discovered attacks, the same list generated by the pin detection sub-component will be used. However, the following filter will be used instead.

If *piece1* is your colour and *piece2* is the opposing colour AND the value of the currently attacking piece is less than *piece2* then keep the element. Otherwise, discard.

Now our explanation software will be able to detect a good mix of common tactics such as double attacks/forks, absolute/relative pins, and discovered attacks.

4.3.4 Move Explanation

The move explanation feature will be one of the most used features within the explanation software. This function will wrap the components that have been discussed so far and gener-

ates an explanation from each one and concatenates it together.

There are three functions that make up this feature. One is the $tacticalExplain$ function, this function will take in a FEN position and a move, then it will process this FEN and pass this along with the move to all the tactical sub-components. Depending on what the sub-component outputs, the tactical sub-components will return either a list of strings or a list of piece lists. For example, the double attack detection function will return a list of elements such as [$piece1$, $piece2$] where $piece1$ and $piece2$ satisfy the requirements of that sub-component. The $tacticalExplain$ function takes these lists and converts lists of pieces into comprehensible sentences using string formatting methods. These sentences are all returned in a list.

The other function is the '$explain$' function. This function is analogous to $tacticalExplain$ and focuses on the strategic component. This function only requires a FEN and uses its mixture of rule-based and scoring-based sub-components to return sentences and scores. The sentences will be kept as is but it is this function's job to convert the correct scores into the appropriate sentence. These sentences are also returned in a list.

The final function that makes up this feature is the '$generate$' function, this function is the function that wraps the previous two together and takes these two lists of sentences and combines them into a paragraph, which will be returned. This is the function that will be called by the API whenever a move needs to be explained.

4.3.5 Static Explanation

The explanation generation feature for moves' primary use case is to explain a move that has just been played. However, we can use the same sub-components but tweak the sentence formation aspect of it to refashion them for another purpose - Static explanation.

This feature idea was inspired by the study feature of Lichess.org, something that it does very well. This feature allows you to import your own games and positions into a dedicated study space where you can analyse them with the engine. The app developed in the project will also have a similar feature where any position can be loaded by the FEN. It would be useful if our explanation generation can be used to identify what the pieces were doing in the loaded position (such as is it threatening something etc).

The current explanation generation will not be satisfactory. That is because most of the current sub-components assume that the FEN and the move will be passed to it in order to calculate the piece interactions from that move. When a position is loaded directly from FEN, there is no 'last move' we can use to pass into these sub-components. Due to this, new sub-components will have to be implemented that will use the tactical sub-components to achieve this new goal.

Attack and Defense
This sub-component will loop through a board and generate a list of all the attacked pieces and all the defended pieces. Due to the relatively small size of the chessboard (there can only ever be a maximum of 64 pieces on a board), this list is generated without any time issues. Now when a piece is queried to see what its role is, these lists can be searched. Both lists contain elements of the form [$piece1$, $piece2$]. The following process is used to generate the sentences:

In the $attacked$ list:
Let k be the queried piece.
For every element [$piece1$, $piece2$] in the list:
    If $piece1 = k$ -> "k attacks $piece2$"
    If $piece2 = k$ -> "k is attacked by $piece2$"

In the $defended$ list:
Let k be the queried piece.
For every element [$piece1$, $piece2$] in the list:
    If $piece1 = k$ -> "k defends $piece2$"
    If $piece2 = k$ -> "k is defended by $piece2$"

With this new sub-component, interactions between pieces can be explained without relying on a move. However, there is a flaw in this sub-component which involves pinned pieces that needs to be solved.

Pinned Pieces
The issue with pinned pieces is demonstrated below.

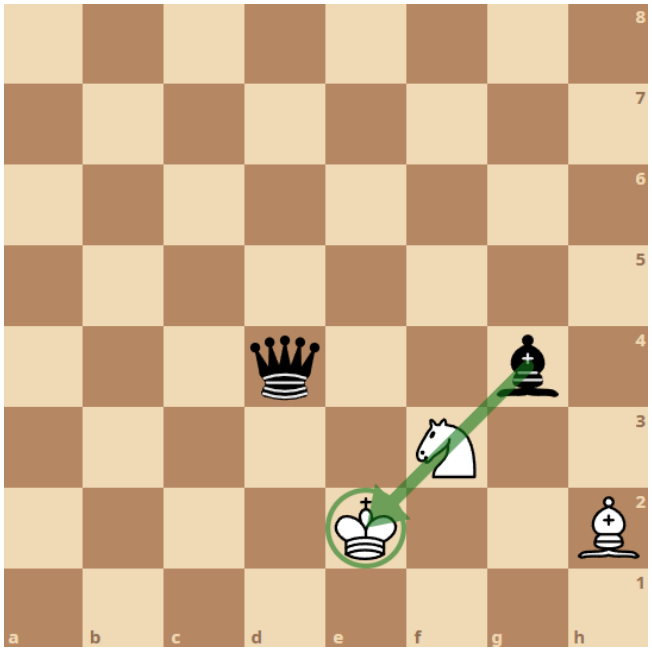

Figure 20: The white knight cannot move because of the pin by the black bishop so it is technically not attacking the black queen or defending the white bishop.

In the above board configuration, the previous sub-component will claim that the white knight is attacking the black queen and defending the white bishop. While these are good points to be aware of, right now it is not technically true. That is because moving the white knight in this position would be illegal as it would put the king in danger. Therefore it cannot do those two things until the pin goes away. This needs to be accounted for in the explanations when the pieces are queried. To achieve this, a similar list will be generated but this time, the elements in the list will have the form: $[piece1, piece2, piece3]$ -> This element represents that piece1

is pinning $piece2$ to $piece3$. Therefore we can now use the following process to generate the appropriate sentences:

In the $pinned$ list:
Let k be the queried piece.
For every element $[piece1, piece2, piece3]$ in the list:
If $piece1 = k$ -> "k pins $Piece2$ to $Piece3$"
If $piece2 = k$ -> "k is pinned to $piece3$ by $Piece1$"

This method of calculating the list of all pins has another use, it can be used to improve the tactical sub-component that discovers pins when a move is made. Consider the following:

Let 'ab' be a move ('ab' = 'e2e4' means a = 'e2' and b = 'e4')
Calculate the '*All Pins*' list when the piece is moved to 'b'.
Calculate the '*All Pins*' list as if the piece were on 'a'.
Return list of elements that are in the *All Pins* list of 'a', but not in the *All Pins* list of 'b'.

This process allows the software to detect when a pin has been broken. It will calculate the list of all pins when a move has been played and looks at the list of the pins before that move was made. If there is a pin that was in the list before that is no longer in the list after the move has been made, that pin has been broken (this can happen several ways, using the [$piece1$, $piece2$, $piece3$] notation used above, a pin can break if $piece1$ captures $piece2$, if $piece3$ moves away or if another piece not as valuable gets in the way of these three pieces).

Square Explain
This is analogous to the '$generate$' function which generates the explanation of a move. It takes in these new sub-components and generates an explanation. The way the function operates is the same as the '$generate$' function but the only difference is the sentences generated will be worded differently to make more sense to the user. The '$generate$' function generates the sentences assuming that moves are being played, while this function will generate explanations assuming that the user just wants to know what roles pieces play in certain positions without necessarily making any moves.

4.3.6 Incorrect Explanations

When learning about a certain topic, be it Chess or History, one of the most effective methods to learn is through testing yourself and correcting mistakes. Therefore, as the primary goal of the explanation software is to help beginners improve chess, the explanation software should facilitate being able to test yourself.

The way the software does this is that alongside the 'generate' function which generates the explanation for the moves, the '$wrong$' function can be requested instead. This wraps around the '$generate$' function. It will take the sentences generated normally and then apply modifications.

The '$wrong$' function will first randomly delete sentences in order to make the explanation incomplete. It will then reverse the keywords at a random. This means 'attacks' -> 'defends',

'loses' -> 'gains' etc. It also reverses colours at a random too, so 'black' -> 'white' and 'white' -> 'black'.

The end result should be an explanation that has a very high probability of being incomplete and/or incorrect. This will give the user the task of trying to correct this wrong explanation after which the user can request the correct explanation to see how they did.

4.4 API Design

As the explanation software only requires FEN and a move as the input, it can be easily integrated into a lot of other applications without many adjustments on either side. A REST API will provide a way for other applications to use our explanation generation software via simple HTTP requests. A REST (which stands for representational state transfer) API uses these HTTP requests to access and use data which can then be used to GET, POST, PUT and DELETE data types. The GET method will be especially useful for us as it will allow us to read the data from our resources. To implement this, Flask was used as it is lightweight and allows users to develop web applications easily. The following code sets up a basic Flask REST API.

```
1 from flask import Flask
2 from flask_restful import Api, Resource
3
4 app = Flask(__name__)
5 api = Api(app)
6 if __name__ == "__main__":
7     run()
8     app.debug = True
9     app.run(host='127.0.0.1')
```

4.4.1 Resources

A REST API treats any content as a resource (A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it). We can easily create a resource that accesses our functions and returns the data we need.

```
1
2 class BestMove(Resource):
3     def get(self, FEN, move):
4         best = engine.returnBest(FEN, move)
5         return {"Current FEN": FEN,
6                 "Move Played": move,
7                 "BestResponse": best}
8
9 api.add_resource(BestMove, "/BestMove/<string:FEN>/<string:move>")
```

Each resource is represented as a class with a get function. This is the function that gets executed when a get request is made to the specified URL. In the above example, this URL is specified on line 9.

The URLs will have the format:
http://127.0.0.1:5000:/Resource/Parameters

So in the above case, it would be the following:
http://127.0.0.1:5000/Resource/<FEN>/<Move>

Now, all that an external application has to do is send a GET request to this address, replace

the <FEN> with the FEN of the current position, and <Move> with the move that was just played. This will return a JSON with all the required information such as a copy of the FEN if that needs to be used and most important and most importantly, the explanation that was generated. We can add similar resources for all the other features that are present in our explanation software. The resource class for the other functions are all very similar in structure to the example above so have been omitted.

4.5 App Development

The app is built using react-native. Expo was used to set up the development environment. Expo is a toolchain built around React Native that allows for the quick development of apps. This was better than the alternative of using React Native init which would require Android Studio or XCode to test the app. There is one more advantage that makes Expo very useful for development and that is live testing. Expo comes with a mobile app that allows live testing. This means changes in code can be immediately seen and tested on an android or iOS device without having to rebuild it every single time. The main disadvantage is that it comes with a lot of integrated libraries which contribute significantly to the file size of any APK file that is created. For example, a simple 'Hello, World' application takes over 20MB of storage when developed with Expo. However, this was an acceptable trade-off for the very fast development productivity it provides.

4.5.1 Layout and Styling

React has a lot of functionality that makes light work of creating app layouts. It has a lot of ready-to-use components such as buttons and it is very easy to create custom components tailored to our needs. We will begin by creating the base of the layout, this will be done using React Native's 'View' component. The 'View' component is the most fundamental component, it is a container that can be styled using CSS. Four Views will be used. One View will wrap everything, then within that View, there will be a View for the Chessboard, a View for the buttons, and, a View for the text display.

The app will be taking a FEN input to load positions. To receive this input, a custom created component will be used. This custom component will consist of a View. The View will contain a Text component, a TextInput component, and two Button components.

This custom component will be linked to the Load button using a React hook after the functionality for the buttons has been implemented.

4.5.2 Chessboard.js

To implement the chessboard, we will need to create a custom component for the board and the piece interactions. Instead of implementing this from scratch, a preexisting one will be used and modified for convenience. The only available component of satisfactory quality uses an outdated Lichess WebSocket communication for its functionality. This will be modified in order to use our explanation engine.

React Native components can take in props, a prop is similar to a function parameter. The main props for Chessboard.js are 'FEN' and '$userColor$'. 'FEN' will take in a string for the FEN of a position and display that on the board. The '$userColor$' will take in a single character, 'w' or 'b'. This character represents from which player's point of view the app displays the board. If

'w' is passed in, it will display the board from White's perspective and if 'b' is passed in, it will be from Black's perspective.

Undo & Reset
These are the simplest buttons to implement, an undo function is built into the library so simply linking the button to this function will suffice. The reset button is also simple to implement, as the library has a prebuilt set method for the FEN. Therefore the reset function consists of setting the game's current FEN to the FEN of the starting position in chess.

Load
This is very similar to the reset button in that we obtain a FEN and then use the FEN set method to set the new FEN to the FEN we obtained. However, this is a little trickier to implement as it required passing multiple variables between different components. To do this we used a react state hook. The way this works is as follows. When the load button is pressed, it triggers a function that changes a state variable. This state variable changes the visibility of the custom input component that was designed to take in the user's FEN input so that it is now visible. Now, when the user inputs a FEN and presses confirm, the state variable is changed once more to make the custom component invisible and the FEN is passed up the component tree into the board component. Now, the function that sets the fen is called and the position is loaded.

Hint
This function plays a computer move. The Chessboard.js had this option but there were two issues. The first one is the Chessboard.js function required the user to play a full game against the computer whereas we only want this function to generate one or two moves when the user is stuck. The bigger problem is that their function did not work. It was using an outdated WebSocket connection.

Fortunately, the stockfish.py script can calculate computer moves using the Stockfish engine and our API has a resource to call that function just for this purpose. Therefore, it was chosen over the prebuilt function which was removed.

Correct & Wrong
The correct button should fetch and display the correct explanation of the move just played on the board. The wrong button should fetch and display the incorrect explanation of the move just played on the board. To do this, the last played move is always kept in a variable. To fetch the required explanation, a URL is constructed from the explanation API's address + what function is needed ($generate/wrong$) + current FEN + last move. A GET request is then sent to this URL and the API will return the corresponding explanation. All that is left to accomplish is to display this explanation which is currently in the board component in the Text Display component. Since they are two unrelated components, adjustments have to be made. The Text Display component is just a View component, to achieve the desired effect, a Text component that displays a JavaScript variable is included inside this View component. This JavaScript variable, which we will call '$explanation$', will be passed in as an additional prop to the board function. We will also pass in a function that sets this explanation variable ($setExplain$) into the board components props. This is then set as a state inside the board component and a new function designed to alter this state is created inside the board component, which we will call $onChangeExplain$.

Now when the explanation from the API is fetched, this state-altering function is called and this calls the $setExplain$ function passed in as a prop which sets the '$explanation$' variable in the Text Display component to the explanation fetched by the API.

Roles
Unlike the other buttons, this button is one that the user should be able to toggle. The toggle is simple to implement with a variable that alternates between 0 and 1. When this button is toggled on, the user should be able to tap any piece on the board and an explanation of what that piece is doing in the position should be displayed. Chessboard.js uses the function '$shouldSelectPiece$' when the board is tapped to handle piece selection. To implement our desired effect, this function is modified so that if a piece is selected then it grabs the square that was tapped and constructs a URL to send an API request to. An explanation is then fetched from the API and using the same '$onChangeExplain$' function that was used for the 'Correct' and 'Wrong' buttons, we can display the explanation on the Text Display component.

Train
Before we implement this button, the function that handles moves ($onMove$) has to be modified to make an API call to our resource which checks for mistakes. Now, on every move, the app will check if it's a mistake and if it is a mistake the resource that the API calls will add that position to the puzzle database.When this button is pressed, it should fetch the puzzle and solution. To do this, a GET request is made to the API which calls a resource that fetches a puzzle from the database where the counter is equal to 0 (so spaced repetition says it needs to be practiced, if there are no puzzles with counter 0 it will look for counter 1 and so on). The puzzle will be in FEN notation, so using the set FEN method it will be displayed. If the user plays a move equal to the solution then the puzzle should be updated in the database (the counter will be incremented by 1).

Now the app will check for mistakes and saves them as puzzles and users will have the ability to train using these individualised puzzles. This completes all the features that we intended to include in the App design.

4.6 Testing

The explanation engine has been tested extensively during development. Likewise with the app. Every time a feature was completed, it would be tested to double check it was working properly. Now that the app is complete, all of it will be tested at once to see if all the features are functioning as intended. I have attached pictures of the tests in the appendix.

5 Evaluation

5.1 Questionnaire

All the features have been tested and working as intended. All that is left is to verify that the app performs its most important success requirement - to help beginners improve their chess ability. In order to measure the effectiveness of the app, a random sample of Chess beginners was used. The average Chess.com blitz rating is reported to be 913 and the vast majority of players are under 1400.

There were 10 participants in the questionnaire, 7 of which were below 1400, classifying them as beginners. The other 3 were above 1500 but under 2000. For ELOs not reported for Chess.com, they were converted using the ChessGoals chart[26]. The questionnaire has three components. One focuses on what methods they used to learn chess, one focuses on the quality of the explanations generated, and one on the quality of the app.

Result Summary

Chess Learning - The vast majority of the 10 picked 'Online Features' as their most useful method of learning (This was specified to be one of the two major chess platforms, Chess.com or Lichess). Only 2 people picked videos as their most useful method of learning Chess with both of them falling in the beginner category. The most quoted online feature was random matches and the second most quoted feature was the tactics trainer. 6 of the 10 participants said they struggled with the middlegame, compared to 4 for the opening. 3 of the 4 who mentioned they struggled in the opening the most fell into the beginner category.8 participants. Positional understanding was the aspect of Chess most people struggled with as 7 of the 10 selected it.

Explanation Generation - For this section of the questionnaire, each participant was shown a series of move explanations and then a series of square explanations. For each move explanation, 7 or more people 'strongly agreed' that it was helpful while 3 or less 'agreed' that it was helpful. 2 or more people 'strongly agreed' that it was accurate and 8 or less 'agreed' that it was accurate. For each square explanation, 1 or more people 'strongly agreed' that it was helpful. 6 or less 'agreed' that they were helpful and in one instance, 3 people did not think it was helpful. The majority of people found the square piece explanations accurate as the most common result was agreed or strongly agree.

Overall, the feedback for the explanation generation was overwhelmingly positive. It was far from perfect however, the comments received usually had suggestions for improvements like implementing additional rulesets or watching out for certain cases. However, there were no complaints or mistakes made by the explanation generation only suggestions for further improvement. The common consensus seemed to be that it was helpful but could be made even more helpful by implementing a certain idea or pattern they had in mind.

App Feedback - The median score for app engagement was 3.5. This is a satisfactory score, most commenters liked the Lichess board style and enjoyed generating explanations as it was a feature they had not encountered in other apps. However, they mentioned the rest of the app is not very engaging, especially the design which was quoted by one participant to look "clunky". For intuitiveness, the mean score was 2. This was the worst quality of the app, for someone with no instruction on how to use it, it would be a struggle to figure out what each button did. This is a design flaw that needs to be corrected for future improvements which will be discussed later. The median score for how frequently they would use the application was 3.5, from the comments received this score was greatly influenced by the previous two questions. While all the participants liked the explanation generation, the lack of engagement in the app and its lack of intuitiveness is what most users said would put them off using it often.

Overall, the feedback on the app was more on the negative side. While there were not many complaints about the functionality itself, the design of the app was not very pleasing to look at or navigate. The project will now be compared to the state of the art before seeing where to go with future implementations.

5.2 DecodeChess Comparison

The only other service that offers automated in-depth chess explanations is DecodeChess. DecodeChess uses a deep learning approach to mimic higher-level cognitive abilities.

Features of explanation that are included in DecodeChess:

- Detects what is being attacked, what's being defended, and undefended pieces.
- Can detect tactical themes such as Fork, Pins, etc.
- Explains Piece Roles at a much higher level*.
- Explains Threats at a much higher level*.
- Explains Plans and Concepts.

*Unlike our explanation system which also does these things, DecodeChess also detects more positional ideas such as good features of the position being threatened and used (such as an open file). It only includes these positional resources if they play a role in the future of the game. While we can easily include a component that detects the positional resources such as open files, diagonals, etc, the way DecodeChess only includes those roles if they matter in the outcome of the game is a level of sophistication above what our explanation system can handle at the moment.

Features of explanation that are included in our explanation system:

- Detects what is being attacked, what's being defended and undefended.
- Can detect tactical themes such as Fork, Pins, etc.
- Explains Piece Roles (but at a lower level of sophistication than DecodeChess).
- Shows tactical threats (DecodeChess also shows positional threats).

Admittedly, the explanation given by DecodeChess outclasses our explanation in most categories. However, even if this is the case, based on the results of the survey we can say that our application is still beneficial for beginners. While the explanation may be inferior, our application does have two main advantages. Our application is less computationally expensive to run. DecodeChess is proprietary, so we have no way of seeing how it is implemented however I imagine that a cutting edge deep learning based method to mimic high-level human cognition will take far more resources to run than the Python based rule detection that is used by our explanation system. Our explanation system is easy to integrate into other applications, all the other application will have to do is send the required API request. Our sub-components all use FEN which is one of the most widely used ways of representing a notation, so most applications will not have to make any adjustments. DecodeChess requires the entire game to be analysed before making any commentary. Our explanation system has no such requirement.

5.3 Future Improvements

DecodeChess gives us a high standard to work towards when it comes to explanation. While the approach used by our explanation system might not be able to reach its level of sophistication, it certainly can be improved. One thing that DecodeChess does that the explanation system does not do is explain plans and concepts. Given more time, this is how these two

areas can be improved.

For explaining plans, there is an open Wikibook of Chess Openings [29] that also explains the plans for the early stages. These pages can be compiled or directly scraped and queried. This way when a player plays '1.e4 e5', our explanation system can query this sequence and fetch any explanations of the plans it finds. While this cannot find the plan in the later stages of the same or from any arbitrary position like DecodeChess can, it will improve the quality of the earlier explanations we provide and is a huge leap in the right direction. To explain concepts, DecodeChess often refers to positional resources such as open files or diagonals. We can implement rulesets to detect these features and then use Stockfish to return a sequence of the next 15-20 best moves. If these features end up being used in this sequence we can include them in our explanation.

Another aspect that requires serious improvement is the app itself. There are several issues with the app. First is the code quality. While there are no performance issues, this app will become difficult to maintain and improve upon. The component based development style of React Native helps this issue slightly, but there are long streams of code in the Board component that can be better split up. The most serious issue, however, is the one reflected by the evaluation survey. The intuitiveness of the application. This can be greatly improved by adding an instructions page that is easy to access and/or by splitting the app across multiple pages and not having everything on one page.

One final aspect is functionality. There is a lot of 'lost' functionality in this project. The biggest example of this is the 'Refute Move'. This is a very beneficial feature and fully functional, however, it was left out of the final app. The reason is that while it worked, its presentation was not polished enough and would have reduced the intuitiveness of the application even more.

Overall, we have met most of the success criteria that we set out to accomplish. The app serves its purpose as a tool for beginners to improve their understanding of chess by providing live explanations. While beneficial, the scope of this project (understanding chess is a vast field) leaves a lot of room for potential improvement. The modularity of the explanation engine itself lends itself quite nicely to this fact as it is easy to add or modify components as needed as new rules are researched and experimented with. One main flaw is that its not very easy to use and is not intuitive. This was the only criteria that we failed to meet.

6 Conclusion

I found this project very enjoyable and difficult at the same time. Every aspect of it has been challenging from project planning all the way to deep technical bugs in a language I was not familiar with. Despite this, I am happy with what I have achieved given that I have met most of the success criteria I laid out. The one that I did not meet was my fault entirely as I had neglected the user interface considerably. My favourite part of the project has been researching Chess and trying to figure out rulesets that can be implemented to get closer to explaining it. Chess is a very interesting game, the more I research it, the more I realise that both DecodeChess and my application have only just scratched the vast world of Chess theory (arguably DecodeChess has scratched a lot more of it than I have). If I were to do this project again, I would improve my planning skills in order to focus on improving the weaker areas of my application such as UI. The evaluation against DecodeChess gave a lot of insight as to what direction this project should go towards but if I were to restart this project - I would like

to experiment with combining both approaches. My application tries to derive explanations from just the position and no other knowledge whereas DecodeChess uses the entire game. If we combine the rule based approach with knowledge of the whole game, we can reinforce certain aspects of our explanation with this new knowledge of what happens later. This would mean giving up our 'On-the-go' generation of explanations and restrict it to only full games, but the level of explanations offered could be far greater.

7 References

[1] Wikipedia Contributors (2019). Chess. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Chess [Accessed 26 Apr. 2022]..

[2] www.ichess.net. (n.d.). Becoming a Chess Grandmaster: The Ultimate Guide. [online] Available at: https://www.ichess.net/blog/chess-grandmaster/ [Accessed 24 Apr. 2022].

[3] Lee, A. (2019). The 20 Highest Selling Board Games of All Time. [online] Money Inc. Available at: https://moneyinc.com/highest-selling-board-games-of-all-time/ [Accessed 27 Apr. 2022].

[4] Shannon, C.E. (1950). XXII. Programming a computer for playing chess. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, [online] 41(314), pp.256–275. Available at: https://vision.unipv.it/IA1/aa2009-2010/ProgrammingaComputerforPlayingChess.pdf.

[5] IBM (2012). IBM100 - Deep Blue. [online] Ibm.com. Available at: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/ [Accessed 25 Apr. 2022].

[6] Wikipedia Contributors (2019). Elo rating system. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Elo_rating_system [Accessed 29 Apr. 2022].

[7] Chess.com (News) (n.d.). How To Quickly Improve Your Chess Game. [online] Chess.com. Available at: https://www.chess.com/news/view/rapid-chess-improvement [Accessed 26 Apr. 2022].

[8] computerchess.org.uk. (n.d.). CCRL 40/15 - Complete list. [online] Available at: http://computerchess.org.uk/ccrl/4040/rating_list_all.html [Accessed 22 Apr. 2022].

[9] The Spruce Crafts. (n.d.). 10 Chess Books to Take You From Beginner to Intermediate. [online] Available at: https://www.thesprucecrafts.com/top-books-intermediate-improvement-611500 [Accessed 25 Apr. 2022].

[10] Chess.com. (n.d.). Best book explaining each move - Chess Forums. [online] Available at: https://www.chess.com/forum/view/chess-equipment/best-book-explaining-each-move

[11] Remote Chess Academy. (2016). The advantage of space in chess. [online] Available at: https://chess-teacher.com/the-advantage-of-space-in-chess [Accessed 26 Apr. 2022].

[12] Chess.com. (n.d.). HEY NOOBS! Forget Openings, Study Tactics (The right way) - Chess Forums. [online] Available at: https://www.chess.com/forum/view/general/hey-noobs-forget-openings-study-tactics-the-right-way

[13] contributor (n.d.). Top 5 Best Chess Training Websites | iChess Blog. [online] www.ichess.net. Available at: https://www.ichess.net/blog/the-5-best-chess-training-websites/ [Accessed 23 Apr. 2022].

[14] Chess.com. (2019). Chess.com - Play Chess Online - Free Games. [online] Available at: https://www.chess.com/.

[15] www.twitchmetrics.net. (n.d.). The Most Popular Chess Twitch Streamers, April 2022. [online] Available at: https://www.twitchmetrics.net/channels/popularity?game=Chess [Accessed 28 Apr. 2022].

[16] Yale.edu. (2015). Transfer of Knowledge to New Contexts | Poorvu Center for Teaching and Learning. [online] Available at: https://poorvucenter.yale.edu/TransferKnowledge [Accessed 25 Apr. 2022]..

[17] Vik-Hansen (Vik-Hansen), R. (n.d.). Pattern Recognition—Fact Or Fiction? [online] Chess.com. Available at: https://www.chess.com/article/view/pattern-recognition-fact-or-fiction.

[18] lichess.org. (2019). The best free, adless Chess server. [online] Available at: https://lichess.org/

[19] The Spruce Crafts. (n.d.). Win at Chess by Knowing Essential Strategies and Tactics. [online] Available at: https://www.thesprucecrafts.com/chess-strategy-tactics-4102130 [Accessed 26 Apr. 2022].

[20] Smolen, P., Zhang, Y. and Byrne, J.H. (2016). The right time to learn: mechanisms and optimization of spaced learning. Nature Reviews Neuroscience, 17(2), pp.77–88.

[21] Tabibian, B., Upadhyay, U., De, A., Zarezade, A., Schölkopf, B. and Gomez-Rodriguez, M. (2019). Enhancing human learning via spaced repetition optimization. Proceedings of the National Academy of Sciences, 116(10), pp.3988–3993

[22] Pyc, M.A. and Rawson, K.A. (2009). Testing the retrieval effort hypothesis: Does greater difficulty correctly recalling information lead to higher levels of memory? Journal of Memory and Language, 60(4), pp.437–447

[23] www.chesspositiontrainer.com. (n.d.). Chess Position Trainer | Chess Training Software - Chess Position Trainer. [online] Available at: https://www.chesspositiontrainer.com/index.php/en/ [Accessed 27 Apr. 2022].

[24] www.chessable.com. (n.d.). Learn chess online: openings, tactics more - Chessable.com. [online] Available at: https://www.chessable.com/about-us [Accessed 25 Apr. 2022].

[25] Anon, (2021). Computers vs Humans in Chess: Who is Better? - Chessable Blog. [online] Available at: https://www.chessable.com/blog/computers-vs-humans-in-chess-who-is-better/ [Accessed 25 Apr. 2022].

[26] Anon, (2020). ChessGoals Home Page - ChessGoals.com. [online] Available at: https://chessgoals.com/ [Accessed 29 Apr. 2022].

[27] Chess.com. (n.d.). The 3 chess phases - Chess Forums. [online] Available at: https://www.chess.com/forum/view/general/the-3-chess-phases.

[28] SQLite (2019). SQLite Home Page. [online] Sqlite.org. Available at: https://www.sqlite.org/index.html [Accessed 29 Apr. 2022]..

[29] en.wikibooks.org. (n.d.). Chess Opening Theory - Wikibooks, open books for an open world. [online] Available at: https://en.wikibooks.org/wiki/Chess_Opening_Theory [Accessed 29 Apr. 2022].

[30] DecodeChess. (n.d.). Smarter Chess Analysis - Start Decoding for Free. [online] Available at: https://decodechess.com/ [Accessed 18 Apr. 2022].

[31] Silman, J. (2010). How to reassess your chess : chess mastery through chess imbalances. Los Angeles: Siles Press.

[32] Chernev, I. and Sloan, S. (2020). Logical chess : move by move : every move explained. Bronx, Ny: Ishi Press International.

8 Appendix

Figure 21: Extreme position tested to see if the explanation engine can spot the tactical ideas.

Figure 22: Explaining the role of the piece in the position. Notice how it is aware of the pin.

Figure 23: Example of a puzzle.

Figure 24: FEN Loader component.