TRAINING BY ERROR:

CHESS TRAINING BY INDIVIDUAL ERROR CLASSIFICATION

Master Thesis

Otto-Friedrich-Universität Bamberg

eingereicht bei:

Prof. Dr. Ute Schmid

vorgelegt von: Christian Massny
Adresse: Kloster-Banz-Str. 2
         96052 Bamberg

Bamberg, 21.04.2013
I. Abstract

Chess Engines have long surpassed all but the most potent of human chess players. The vast majority of chess players cannot even come close to beating any given engine on a personal computer with mainstream hardware.

Hence chess programs are mostly used for two purposes: analyzing past games and getting used to playing new openings. The analysis is mostly done by hand, importing any past game and the replaying it with the calculations of the engine showing when mistakes were made in the game.

The goal of this thesis now is to analyze past games on a larger scale using the complete game database of a particular player. The software to be programmed will import any number of games and analyze them. The analysis will find mistakes made by the player using an open source chess engine and then classify the mistakes the player made so that statistical information regarding the type and magnitude of mistakes the player makes will be available for further training.

Each mistake can be singled out and analyzed individually. At the end of this thesis the author will propose a way of using the error database to teach the player to get rid of their mistakes even better.
II. **Eidesstattliche Erklärung**

Hiermit bestätige ich, dass

- die vorliegende Masterarbeit selbständig durch den Verfasser und ohne Benützung anderer als der angegebenen Quellen und Hilfsmittel angefertigt wurde,

- die benutzten Quellen wörtlich oder inhaltlich als solche kenntlich gemacht wurden; und

- diese Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungskommission vorgelegt wurde.

**Bamberg, 21.04.2013**

_____________________________

Christian Massny
III. Acknowledgements

This work would not have possible without the support of many other people.

First and foremost I want to thank my wife for letting me take my paternal leave. Without her I would not be writing this thesis. As I am writing this she is handling the kids so that I can finish my work.

Second I want to thank Prof. Ute Schmid, for letting me write a thesis on a subject that really is right up my alley. Also I want to thank her for the discussions we had on my thesis which really helped shape the way I could get my grip on the topic.

The program itself was built using Open Source components so I want to thank Nazmi Altun for the PGN Viewer, Stephan Hoedtke for the UCI Chess Trainer, Pawel Idzikowski for the SharpSerializer Library and of course Ralf Schäfer and Volker Böhm for the Spike Chess engine.

Last I want to thank JJ Cale for his soothing music which helped me a lot to get my head around the trickier part of the programming.
**IV. Index**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Abstract</td>
<td>I</td>
</tr>
<tr>
<td>II.</td>
<td>Eidesstattliche Erklärung</td>
<td>II</td>
</tr>
<tr>
<td>III.</td>
<td>Acknowledgements</td>
<td>III</td>
</tr>
<tr>
<td>IV.</td>
<td>Index</td>
<td>IV</td>
</tr>
<tr>
<td>1</td>
<td>Chess and Artificial Intelligence</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Playing Chess</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Playing Against The Computer</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>Advancing AI Chess Training</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Chess AI and training chess</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>The game of chess</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Artificial Chess Programs</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Human Chess Training</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>The AI Trainer</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Analyzing chess game databases</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Algorithms</td>
<td>19</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Parse File</td>
<td>21</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Analyze Game</td>
<td>21</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Find Mistakes</td>
<td>23</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Extract Features</td>
<td>23</td>
</tr>
<tr>
<td>3.1.4.1</td>
<td>Opening</td>
<td>26</td>
</tr>
<tr>
<td>3.1.4.2</td>
<td>Middle Game</td>
<td>27</td>
</tr>
</tbody>
</table>
3.1.4.3 End Game
3.1.5 Statistics

3.2 Program Structure

3.3 Pre-existing components

3.3.1 PGN Viewer
3.3.2 UCI Interface
3.3.3 Chess Engine
3.3.4 SharpSerializer

3.4 Classifier Architecture

3.4.1 Mistake
3.4.2 Classified Game
3.4.3 Game Database
3.4.4 Statistic

3.1 User Interface

3.1.1 Main Window
3.1.2 Game Selector
3.1.3 Statistics Window
3.1.4 Error Display

4 User Study

4.1 Methodology
4.2 Questionaire
4.3 Results

5 Discussion
5.1 Suggested Improvements to the Prototype .............................................................. 47
5.2 Future Studies ........................................................................................................ 48

6 Appendix .................................................................................................................. 49

6.1 Bibliography ........................................................................................................... 49
6.2 Table Index ............................................................................................................. 52
6.3 Figure Index .......................................................................................................... 53
6.4 User Study Tasks .................................................................................................. 54
6.5 Questionaire – German Translation ....................................................................... 58
6.6 Questionaire – Answers ........................................................................................ 62
1 Chess and Artificial Intelligence

1.1 Playing Chess

“When Sissa had invented chess and produced it to King Shihram, the latter was filled with amazement and joy. He ordered that it should be preserved in the temples, and held it the best thing that he knew as a training in the art of war, a glory to religion and the world, and the foundation of all justice.” - ibn Khallikan, thirteenth century [from Shenk 2006]

There are hundreds of myths how chess was created and it is not unduly named the “Game of Kings”. Chess has fascinated the best and brightest of every era from every occupation. Stefan Zweig’s Schachnovelle was inspired by the game, as were the beginnings of artificial intelligence. Poets and scientists alike gain insights into their subject of study by studying the 64 black and white squares that make up the chess board.

Chess is of great interest for psychologists ([Gobet, Jansen 2005], [Bönsch-Kauke 2009]), sociologists ([Mazur et al. 1992]), mathematicians ([Morse, Hedlung 1944], [Fraenkel, Lichtenstein 1981]) and of course computer scientists ([Turing 1950], [Minsky 1961], [McCarthy, Hayes 1968], [Littman 2012]). So it is safe to say that chess was and still is fascinating subject to learn about the mechanics of games as an important part of human society.

The goal of this work is to assess where AI chess stands as a competitor and companion for human chess players and provide some insight where computer may help in the future to advance players in their understanding of the game.

1.2 Playing Against The Computer

One of the top research areas in artificial intelligence has always been chess and artificial chess players. Chess is complex to master but its basic rules are few and easily implemented. The “Game of Kings” has both fascinated and inspired humans for centuries. So when science began to play a bigger role in our model of the world during Rationalism and the Age of Enlightenment, people were trying to build chess playing machines. Today we might consider the Chess Turk to be a scam, but during that time it was a wonder that fooled the most literate and intelligent people of the time. Among them were such names as Mark Twain [Buchanan 2005].
So when scientists began to realize that with the availability of computers artificial intelligence could be more than a man hidden in a box, chess was of course one of the areas where computers could be tested against the human mind they tried to imitate. Having this in mind there is no wonder that Alan Turing himself is considered to be the first to write a chess playing program or better algorithm. Together with his colleague David Champernowne they created a chess-playing “program” known as Turochamp [Copeland 2004, p. 563]. Turing did not have the infrastructure to implement the program but he and his students created an algorithm that plays chess against a human player.

As computers became more commonly available, more and more scientists began developing better and better algorithms for chess playing computers. Of course there was not only scientific interest in finding out how good a computer chess program could become: Among many other smaller enterprises IBM saw the possibility to promote the power of its computers by promoting a tournament between the best AI chess program of that time, Deep Blue, and the reigning Chess World Champion Garry Kasparov. The tournament got featured in almost every news channel imaginable and although Kasparov played at his best and fought hard to get the upper hand, Deep Blue closely won the tournament.

This, of course, marks a big leap for chess programs and artificial intelligence alike. Today’s mainstream computer chess programs, which can be bought for less than 100€, can beat almost any human player when run on a home computer as can be found now in almost every household.

Programs such as Deep Junior or Deep Fritz now also master the hard to grasp concept of positional play, where the player does not play to capture pieces in order to gain an advantage but merely tries to position their pieces just a slight bit better with each move. Even more so these programs now achieve what can only be called playing beautifully [Bushinsky 2009].

So with the scholar having long surpassed the teacher, it is safe to say that – at least for AI chess – the singularity has come. There is no need to invent better chess programs, the advancement of computing power alone will advance them,

---

1 The singularity is “. . . the imminent creation by technology of entities with greater-than-human intelligence.” [Vinge 1993]
and since there is no human left, who can beat any of the top chess engines, it is time for the scholar to really become the master.

### 1.3 Advancing AI Chess Training

Chess programs are already used to train human players. But they cover only a part of what a human trainer does and do not make use of what the computing power of today's hardware really can offer apart from playing chess. Still, almost all of the chess training when using a computer is being done on a single game basis. Computers help us find and analyze those games but we still look at each game of a time. Computers help us to try and find interesting games in vast databases but the player still needs to ask the right questions. Mining these databases is the key to helping human chess players in a way only computers can do. We need to feed the databases more information to find anomalies, trends and insights not only from a snapshot of a player but their whole past game history.

The framework and software presented in this work will not only analyze a set of games and tell the player who lost what game. It will extract mistakes even from successful games and provide the chess player with additional information on these mistakes. The program will automatically find mistakes, extract their features and provide statistical information in a way that will make it easier for players to find their own and their opponents' mistakes.

The second chapter will start with a few prerequisites the reader needs to understand the following sections of this work. It will explain the game of chess itself, as far as this is possible within the boundaries of this work. Also it will present a few things the reader needs to know about competitive chess playing. This chapter will also present the theoretical foundation upon which the idea of the software and the analysis framework is based. After that there will be a presentation of the framework of features and analysis that is used in the software and will create the insight into the game database.

The third chapter of this thesis then will explain how the software works and of which components it is composed. The main focus here will be on the algorithms used to extract and analyze the mistakes from the game database.
A user study consisting of amateur chess players from a local chess club and an internet chess site will judge the usefulness and potential of this idea and software in chapter 4.

The last chapter will discuss both the results from the user study as well as possible improvements and future additions to the framework.
2 Chess AI and training chess

2.1 The game of chess

For the understanding of this work it is essential to have an understanding about the basic concept of chess. The reader does not need to be able to play chess to understand this, though, of course, it helps. Additionally, the reader needs to know a few things about competitive chess players (such as the ELO ranking system) and chess tournaments. I will explain these concepts in this chapter. If you are already familiar with them, you may skip to the next chapter.

The chess board consists of 64 squares alternating in a black and white pattern. Two players sit at the opposing sides of the board and each has a set of different pieces. One player’s pieces are of white color, the other’s black. Although for aesthetic reasons the actual colors of the pieces and board may vary, the colors and squares are always referred to as black and white.

![Chess Board - Initial Position](image-url)
The pieces have different abilities to move around the board. The figure below shows these abilities:

![Chess Piece Movements](image)

The pieces of chess are (from left to right and top to bottom): the Pawn, the Bishop, the Knight, the Rook, the Queen and the King. The aim of the game is to challenge (check) the opposing king in such a way that he cannot move onto another square that is not challenged. This is called check mate. The figure below illustrates a few possibilities for check mate. In each position it is black’s turn and white just checkmated the opposing King.

![Check Mate Positions](image)

The chess pieces are considered to have an exact in relation to the weakest pieces, the Pawn. These values are derived from their movement ability. These values are particularly important when analyzing a position on the board and evaluating
which player is to be favored. The favorability of a position is valued in Pawns, too. The table below shows the values of the pieces as they are told the novice player and as scientific analysis of thousands of games revealed.

<table>
<thead>
<tr>
<th>Piece</th>
<th>Value as taught</th>
<th>Value as computed²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pawn</td>
<td>1</td>
<td>0.196</td>
</tr>
<tr>
<td>Knight</td>
<td>3</td>
<td>0.37</td>
</tr>
<tr>
<td>Bishop</td>
<td>3</td>
<td>0.468</td>
</tr>
<tr>
<td>Rook</td>
<td>5</td>
<td>0.569</td>
</tr>
<tr>
<td>Queen</td>
<td>9</td>
<td>1.135</td>
</tr>
<tr>
<td>King</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Chess Piece Values

Chess is about competition. So naturally there are a lot of chess tournaments around the world. The most well-known is, of course, the World Chess Championship, where the world’s best chess players compete for the title “World Champion”. At the time this work was written Viswanathan Anand has been the World Champion for 3 years and will try and defend against his 2013 challenger Magnus Carlsen. Chess Champions are chosen like Boxing World Champions. The current holder of title has to beat a challenger at regular intervals (once a year) to be able to continue to name himself world champion. The challenger has to win the candidates tournament in order to gain the right to challenge the champion. The candidates are selected from the top ranking players in the world.

Chess players are ranked using the ELO Rating system. Its inventor, the physicist Arpad Elo, proposed the ranking as a statistical approximation of the supposed playing strength of a chess player. Using this rating one can make statistical assumptions on which player should win a game using the formula:

$$E = \frac{1}{1 + 10^{-(R_A-R_B)/400}}$$

where $E$ is the expected result of the game, with 1 being a win, 0.5 a draw and 0 a loss, $R_A$ and $R_B$ the players’ ratings. This means in short that if a player has a rating of 200 ELO-points higher than their opponent the chance for winning is 0.76.

² Learned values from [Mannen, Wiering 2004]
The rating of a player is updated after a tournament using the formula:

\[ r_{post} = r_{pre} + K(S - S_{exp}) \]

Where \( r_{post} \) and \( r_{pre} \) are the ratings after and before the tournament, \( S \) is the total score the player achieved, \( S_{exp} \) the expected score using the formula for winning expectancy from above. \( K \) is a magnification factor depending on how fast the chess organization wants the players’ rating to be subject to change. If the player has played less than 20 games, a provisional rating is used compiled from the average rating of their opponents [Elo 2008]. The current rating of the reigning World Champion is 2783, really good chess players can be found beyond the 2000 ELO margin and the average rating for chess club players is 1500. So how can one reach such a high level of expertise?

„I think that the analysis of one’s own games is the main means of self-improvement” – Artur Yusupov

A chess player can reach that average rating of 1500 ELO without training much else than learning the principles of chess and having a basic repertoire of opening moves. After that it is all about game analysis. Players analyze games of Grandmasters, their opponents and of course their own. The best chess trainers in the world agree in one thing; if a player wants to get better they need to analyze their and others’ games [Dvoretsky, Iușupov 2006].

A lot of this training is done using the help of AI chess engines. The next chapter will explain how these engines work and what they are capable of doing.
2.2 Artificial Chess Programs

As laid out in the chapter before, artificial chess programs have long surpassed the average chess player and only a handful of players in the world can win against a current chess engine on a moderately powerful modern computer. So how do these programs achieve such strength in play and even play more beautifully than in many draw-drenched human tournaments? This chapter will give a brief overview of how chess engines generally operate. Of course I do not have insight into trade secrets of commercial chess engines, but some of the most powerful engines of today are open source and thus can be used for analysis.

Basically each chess engine consists of two parts. First, the positional analysis of the current situation of the board determines which of the players is to be favored with the current position of the pieces on the board. Second, the search algorithm probes different move combination to try and improve this analysis.

Like Google’s Pagerank algorithm [Brin, Page 1998] this is a fairly straightforward concept but presents the problem of scale. Although, with the available computing power growing exponentially, games like checkers have been solved [Schaeffer et al. 2007]. This has not been done for chess yet. Why is that? The board is the same, so the number of moves should not be all too much higher – at least for a computer. Unfortunately this is not the case. At any given position there are at average 30 legal moves to make, a typical game lasts around 40 moves. This still sums up to $10^{120}$ positions to be considered before the first move can be made. If you would like to store all possible positions in a lookup table and then just add the best move – however computed – to the table this would be $10^{43}$ positions [Shannon 1950]. The current storing capacity on our whole planet is in the exabyte range. As of 2007, scientists calculated them to be 295 exabytes to be precise. In the timeframe analyzed the storage capability grew 58% per year [Hilbert, Lopez 2011]. If we take this growth, consider it to be constant and leave out physical restrictions, the earth would have enough storage for this lookup table to be stored in 125.845 years, but it would have to erase every bit of data stored.
So, a lot of thought has gone into techniques for cutting off branches of the game tree. These concepts include opening books, Alpha-Beta-Pruning, grandmaster databases and the evaluation of the current board. I will explain these concepts in the following section, using the specificities known of the Deep Blue chess engine.

One of the most important parts a chess player has to learn is the opening. Mostly, learning in this case means learn by hard. Over the centuries chess has been played, but mainly in the 20th century, players systematically evaluated and improved the opening moves in a chess game. Now, if a player wants to play competitively, they have to pick a set of openings from the vast possibilities which suit their playing strength and style and then learn them, their variations and patterns by hard. The same was done while building Deep Blue. A grandmaster of chess helped the programmers pick openings that Deep Blue handled well, which at that time meant, positions that had complex tactical possibilities. On top of that, he selected openings that he considered likely to arise in a game with Kasparov.

None of these were played in the matches, though [Campbell et al. 2002].

Although techniques, such as opening books, try to limit the computing necessary when playing chess, the task at hand still mainly is performing a search. Find the best possible move from the available moves at the current status of the game. The problem both humans and computers face is, that the further down you go, the possible move combinations grow exponentially and the amount of time for finding the best move is limited. From each move there are numerous possible successors so that the tree with all possible moves branches out very wide and deep. On top of that the search has to go both wide and deep into the game tree. A move that looks good in a wide search might turn bad if the search goes deeper. On the other hand a deep search might overlook a move that does not look good at first hand but might be a good move when searching deep. One particularly popular algorithm in this context is called Alpha-Beta-Pruning. Its purpose is to cut leaves off the tree early so that the search can concentrate on moves more likely to produce good results. Alpha-Beta-Pruning is a depth first search algorithm.
The following figure shows the algorithm in pseudo code.

```
Integer alpha-beta(chessboard b, int alpha, int beta)
{
    determine successor positions p1,..pn (possible moves);
    if (depth == 0) return bruteForceSearch
    else
    {
        m = alpha;
        for (int i = 1; i<depth; i++)
        {
            t = -alpha-beta(p1, -beta, -m);
            if (t > m) m = t;
            if (m >= beta) break;
        }
        return m;
    }
}
```

Figure 4: Alpha Beta Pruning Algorithm [Knut, Moore 1975]

The algorithm uses both upper and lower bounds and is an optimization of the Minimax-Algorithm. It introduces lower (alpha) and upper bounds (beta) for the decision if a leaf is worth analyzing or if it should be removed from the search. The algorithm minimizes the result for the other player so that alpha is the result that the analyzing player can achieve minimally (and thus should be maximized) and beta the result that the opponent can achieve when playing optimally (and thus should be minimized). The algorithm starts with the first leaf and save the optimal opponent move as the new lower and upper bound. Then the algorithm goes to the next leaf and checks the first opponent move. If the evaluation of the move is lower than beta it will be saved into beta. Then the algorithm cuts this leave off the tree as the player will never play a move that is worse than the one already analyzed. If the evaluation is higher the search will continue at that leaf and cut off the one first analyzed. The graphical representation of the algorithm can be found in the figure below.

Figure 5: Alpha-Beta-Search tree after 4 half-moves [from Ertel, Mast 2011]

---

1 For further information on the Minimax-Algorithm see [Campbell, Marsland 1983]
On top of that, Deep Blue used a database of over 700,000 games played by grandmasters to help the search in situations where the game has gone off track from the more rigorous opening book. As this database does not simply tell the best moves, but instead suggests statistical advantages of moves played by a bigger portion of the grandmaster database, it does not directly command a move to be made, but merely adds an offset to a favorite move which is then added to the general search. So, a move that is and was “popular” among chess grandmasters gains an edge in situations where several moves are considered more or less equal by the search and evaluation algorithms. Many features are taken into consideration to generate the offset. [Campbell et al. 2002]

- Absolute number of times the move has been played
- Relative number of times the move has been played in relation to other moves
- Player strength that chose the move
- Recentness of the move
- Results of the move
- Commentary of the move (game commentaries often mark good moves with an exclamation mark, bad ones with a question mark)
- Commentary moves (commentaries often propose alternative moves, commentary moves are considered worse)

The last part of chess engines, I want to explain, are the evaluation functions. The concept for this is directly derived from human chess players. When facing a particular position at a chess board, chess players are taught to run and use a certain checklist to evaluate the position and find strengths to use and weaknesses to exploit or be aware of. The checklist consists of points that differ much in complexity and their importance differs over the course of the game. Easy to describe and calculate is the check of the chess pieces. It is agreed among chess players that each piece should be guarded once more than it is attacked. If it is not, it is considered to be a weakness. The reason for this is that if the opponent challenges a piece that is attacked and guarded with another piece, the player has to react to that threat. More complex concepts are how well the pieces play together, how good the pawn structure is, and who occupies which portion of the board and a few more. Chess engines’ evaluation functions work the same way.
information for each feature on the checklist is enriched with an evaluation value and then all features are added together summing up to the evaluation of the position on the board [Campbell et al. 2002].

2.3 Human Chess Training

Since the methods of playing chess do not differ all too much between humans and computers, humans learn playing chess in a very similar way as do chess engines. Of course there is a major difference for humans. We cannot consciously think through millions of possible moves in the fraction of a second. The human brain still has an edge over even today’s most powerful computers in two fields: parallelism and pattern recognition [Fahle 1994].

So human players, too, learn opening books but do not learn each and every one of them. They just learn those, that fit their playing style. But moreover they learn patterns. Rather than memorizing the moves that lead to a certain position, they memorize desirable patterns and in which kind of situations they occur. So, when a memorized situation occurs the search tree is cut dramatically to only those moves that might lead to that particular pattern [Gobet, Jansen 2005].

In their 2005 paper “Training in chess: A scientific approach”, Gobet and Jansen not only concentrate on learning openings, but rather propose an integrated theory of how and what a chess player needs to train to advance from being a proficient competitive player to being an expert at the game. They categorize these two types of players with their ELO number. While the former has an ELO of around 1800, the latter is set to play at a level of 2400. Grandmasters are considered to be in the 2600-2800 ELO range whereas the average player has an ELO of 1500. Although this paper concentrates more on these average players than on the experts, their findings can be used here, too, as the same principles apply. Only the level of expertise differs.

Gobet and Jansen consider three dimensions to be relevant in learning chess: Type of encoding, diachronic dimension and chess contents. The figure below illustrates these dimensions.
The encoding differentiates between implicit and explicit encoding. Identifying explicit knowledge about the game of chess is quite easy. A simple web search will uncover much of this knowledge such as openings, endgame patterns, checkmate patterns, desirable piece configuration et cetera. Most of this knowledge needs to be learned by hard. At least up to a certain point, as the player needs a basic set of patterns and openings to be able to understand higher levels of abstraction. The player needs to find a balance between knowing the right move and knowing how to find the right move in a situation.

Implicit knowledge is a lot harder to describe and achieve. Implicit knowledge in chess game is often referred to as intuition. An experienced chess player just knows which are most likely advantageous moves in a position and will intuitively limits their search for a good move to these candidates. Gobet and Jansen suggest that implicit knowledge can be acquired by using computer chess programs to analyze and play around with certain positions and thus learn to recognize good moves for a certain pattern. In my experience as a competitive chess player, playing games with very limited time for a move, such as blitz chess, where both players have only five minutes each for the game, trains this intuitional play. The reason for this is that the depth of the search is limited dramatically. What also proved effective, although not well liked by traditional trainers, is a variation of
chess, called “bughouse chess” \textbf{[Bughouse Chess 2013]}. In this chess variant two player teams play against one another with one player playing the white pieces and the other playing the black ones. When one player captures a piece of their opponent the team-mate can place it on their board wherever they see fit. This trains intuitive awareness of check mate patterns.

\textbf{Diachronic Dimension}

The chess game divides into three phases: opening, middle game and endgame. Each of these phases has specific characteristics and needs to be learned and taught in a different way.

The opening of the game is mostly rote learning, where a player tries and memorizes a set of openings and their variations that fit their playing style. Most players only learn a subset of the huge variations that are possible within the first 10 to 15 moves of the game. Still, even with the reduction of the “pre-computed” part of the game tree that needs to be learned, they have to find a balance between memorizing and understanding an opening. Only if a player truly understands an opening and its ideas and motives they can quickly react to variants and the transition to the middle game is easier.

Since the search tree branches out quickly, in the middle game it is much harder to learn anything by hard. So most players learn a pool of patterns and maneuvers from which they can derive their plans in the middle game. Examples for these are pawn advances, attacks on the king, regrouping of pieces and transitioning to a favorable endgame. All these can be memorized but need a good understanding of their principles in order for the player to be able to apply them to an actual game.

The last part of the chess game is the endgame where only a few pieces are left and players mostly try to either transform a pawn to a better piece by moving all the way to the opponents “endzone” row or they aim for a check mate. Still with fewer pieces on the board there is more space to maneuver and more possibilities for the pieces to move. So players learn how to play certain piece combinations. One of the most common of these are Pawn, Rook endgames where the rooks on the files try to guard the advancing of the pawns to the “endzone”. Again players learn pattern and movement maneuvers such as having the same position as before, only it is the other player’s turn \textbf{[Gobet, Jansen 2005]}. 
Chess contents

The last dimension Gobet and Jansen define, are the two domains of tactical and strategical play. Tactical play tries to find combinations of moves that result in capturing one of the opponents pieces or checkmating the opponent. Strategical play tries to improve the positioning and connectedness of the pieces little by little so that these small advantages add up to a big one in the future. This advantage itself is behind the search horizon of the player.

Again players need to acquire patterns and maneuvers and learn to transfer them into real life chess games. Tactical play is easier to learn as strategical, as the goal in hand easy is to formulate and the effect and success of the maneuver is easily recognizable. Strategical play on the other hand is harder to teach as the effects may not occur until the endgame. Gobet and Jansen suggest to learn this concept alongside with the player's opening repertoire as the goal of many openings is to get into a strategically advantageous position.

Summing up these aspects one can safely say that the field of learning and teaching chess is vast and can hardly be mastered alone. The next chapter will shed light on how AI chess program help chess players improving their strengths but also what they are lacking at the moment.
2.4 The AI Trainer

Chess Players are already using the superior playing strength of computer chess programs for training purposes. They use the fast analytical insight the program has to replay their own or their opponents games move per move to search for weaknesses.

They also use the engine to train newly learned openings. They lock it so that it only plays a particular opening in order to get used to it and gain match experience against a well playing opponent.

In addition to this you can try maneuvers against the computer that you are not quite sure of. The computer does not care if it wins and so you can go back to the original situation and try to find an alternative solution. Computer chess programs also give the players access to the history of chess games. They can replay the games of all the masters of chess and their most beautiful and hardest fought games.

AI programs also try to simulate players of lower strength by limiting the computing power or not always chosing the best move. There are efforts to try and emulate the playing strength of different levels of players by learning their behavioral pattern with Support Vector Machines for example, but these experiments have not proven all too successful [Paulsen, Fürnkranz 2010].

Still, these training possibilities do not make use of everything databases can offer the user. While most databases and programs let you search through games using facetted search and other powerful search tools they don’t offer so much on the analysis of games. In the next chapter I will explain how this work will try and offer the player more advanced analysis possibilities on a larger number of games.

There are different definitions for classification in computer science depending on the field of study and the application of the classification results. Within the machine learning domain classification means:

“At its broadest, the term could cover any context in which some decision or forecast is made on the basis of currently available information, and a classification procedure is then some formal method for repeatedly making such judgments in new situations.” [Michie et al. 1994, p.2].
In this context we will use the definition from the domain of software engineering, defect analysis to be precise. In their 1992 paper Chillarege et al., use the features extracted from a defect to try and find statistically relevant information so that the defects can be classified [Chillarege et al. 1992].

A similar approach will be used in this paper. The algorithm will extract features from the mistakes found in the games and present them to the user in such a way that they can derive a classification or tendency from the statistics.
3 Analyzing chess game databases

The following chapter will introduce the program which has been developed for the analysis of the chess games. First I will present the algorithms used in the program and give a short overview of the structure of the program. The last part of the chapter will explain the user interface of the program.

3.1 Algorithms

Figure 6 shows the algorithm which is used for analyzing the games, extracting the mistakes and their respective features and finally outputting statistical information regarding these features.

First the user selects the file which contains the games that are to be analyzed. The software parses (see 0) this file expecting the Portable Game Notation (pgn) format. It extracts the game notation itself as well as tagged information about where the game was played, which players played the game and other relevant information. Then the program searches for a recurring player which is used for analyzing the games. Only the mistakes of this player will be output, mistakes of other players will be discarded.
Then the software starts analyzing with the first game in the file. Its positions will be analyzed (see 3.1.2), and then the mistakes will be extracted (see 0). For each mistake its features will be gathered and saved into the database (see 3.1.4). This then is repeated for each game until all games within the file have been processed. Lastly the games will be analyzed for statistically significant information regarding the nature of the mistakes made in the collection. This information is then presented to the user and can be used for further analysis.

Figure 7: Complete Algorithm
3.1.1 Parse File

The algorithm for parsing the file is straightforward. It reads the data and then checks for syntactic consistency regarding both the game information (tags) and the notation of the game itself. If everything is correct, the game is read and then put into the database for further processing.

![Algorithm - Parse File](image)

3.1.2 Analyze Game

The program uses a UCI-compatible engine to analyze the game and its moves. These engines transmit information regarding the suggested moves and provide an evaluation about the desirability of the position on the board. This information is translated to piece benefits and measured in centipawns. If a player has an evaluation of 150 centipawns in favor of his color, it means that the advantage the player has, is the equivalent of having one and a half pawns more on the board than the opponent.

For analyzing the game, the software starts with an initialized board and then transmits each move to the engine. The engine is then given a signal to search for
the best move in the situation. It then transmits the evaluation value via the command line interface. After a given amount of time the program sends a stop signal to the engine. It then extracts the evaluation for the last, and therefore best, move transmitted and attaches the information of the evaluation to the move. The algorithm repeats this procedure until the game has ended.

Figure 9: Algorithm - Analyze Game
3.1.3 Find Mistakes

As mentioned before, the software only searches for the mistakes of one particular player. When trying to find mistakes in the game it therefore only analyses moves made by that particular player. If the move color is correct it compares the evaluation of the move with the one of the previous move. If the evaluation is worse than the one of the previous move and the difference is bigger than a threshold the move is marked as a mistake and saved into the database.

This procedure is continued until there are no more moves in the game.

![Algorithm - Find Mistakes](image)

Figure 10: Algorithm - Find Mistakes

3.1.4 Extract Features

After the mistakes have been saved into the database, the program extracts their features and saves them for later statistical analysis.

First the move information is extracted. The move information contains the color of the moved piece, its source and destination field, as well as the number of moves.
that have been made in the game so far. The move information also covers the notation of the move, the gravity of the mistake (in centipawns) and a representation of the board.

Additionally the gravity is used to categorize the mistake into one of five gravity classes: 100-200, 200-300, 300-400, 400-500, 500+.

The next feature is the quadrant to which the piece moved. The quadrant can be one of four possibilities: white kingside, white queenside, black kingside, white queenside.

Then the information regarding the opening that has been played in the game is being extracted. Chess Openings are ordered by using the ECO Code System [Chess Informant Inc. 2013]. The ECO Code consists of a combination of a letter followed two digits. The letter classifies the kind of opening (e.g. Open or King Pawns Games) while the digits represent a certain opening or variation of an opening. This information is collected and the letter is stored individually for later analysis. For the same intent, the program extracts a string representation of the first three moves and stores them into the mistake database.
The most complicated feature to extract is the phase, in which mistake was made. The phases are a fuzzy concept and blend into one another. Chess players will mostly agree which phase a game is in, but only mostly. That is why the program tries and extracts probabilities for each game phase. The thresholds for determining these probabilities were found in discussions with other chess players. Since the algorithms for the extraction of the phases are more complex, they will be explained individually for each phase in the following three subchapters.

After the extraction of the game phase all the features of the mistake are stored in the game database.

Figure 11: Algorithm - Extract Features
3.1.4.1 Opening

The program determines the phase of the game by taking two features into consideration, the number of moves made and the number of pieces still on the board.

The first ten moves of a game are almost always still the opening phase, so the program considers these to be surely opening and thus sets the probability to one. If more than 15 moves have been made, the opening is surely over and so the probability is set to zero. If the move count is 11 to 15, the program then looks at the number of pieces still on the board. If there are more than 28 pieces still on the board, the game surely is still in the opening, if there are 20 or less pieces it surely is not. If it is between these values, the program first calculates two different possibilities using the move and piece count as shown in the figure below. It then takes the one probability that is bigger and saves it into the mistake information.

![Figure 12: Algorithm - Extract Features - Opening](image-url)
3.1.4.2 Middle Game

The algorithm for determining the probability for the middle game phase is similar to the one for the opening phase. But of course, it uses different thresholds.

If a game has 10 or less moves made or more than 40 moves have been made, the game is surely not in the middle game phase. If the move count is from 11 to 40, the program uses the piece count for further clarification. If there are more than 28 pieces still on the board, the game is still in the opening and thus not a middle game. If the piece count is in the range from 17 to 28 (close to opening) or 16 or less (close to end game) the probability is calculated as shown in the figure below.

![Algorithm - Extract Features - Middle game](image-url)

Figure 13: Algorithm - Extract Features - Middle game
3.1.4.3 End Game

The last feature to extract is the probability for the end game phase. If there are more than 40 moves made, the game surely is an endgame; if it is 10 or less is surely is not. After that the algorithm’s focus moves on to the piece count. If there are more than 20 pieces still on the board, the game is not in the end game phase. If there are 12 or less, then it surely is. For piece counts in between these thresholds the probability is calculated as shown in the figure below.

![Flowchart](image-url)

*Figure 14: Algorithm - Extract Features - Endgame*
3.1.5 Statistics

Since the program is designed to handle big chess databases the statistics are stored in hash maps. The algorithm selects a feature and then generates a string representation of the feature, the key. The feature is then stored into two hash maps; one solely counts the members, the other stores the mistakes for that key. The algorithm then sorts the hash map by converting it to a list and saves the top 5 keys into a shorter list which can be used more efficiently. This procedure continues until all features have been processed.

![Figure 15: Algorithm - Statistics](image-url)
3.2 Program Structure

The program mainly consists of three different components. The program itself and its forms and visual components, the chess library, which interacts with the chess engine and stores all chess specific components such as boards and pieces, and the classifier, which provides all functionality for analyzing the chess games and their components.

3.3 Pre-existing components

The program makes use of a set of pre-existing components which were used to speed up the prototyping process and made it able to focus the programming effort on the main task of finding and analyzing mistakes.

3.3.1 PGN Viewer

The PGN Viewer is a piece of software programmed in C#, which provides functionality for importing and viewing PGN documents. PGN is a standard for digitally representing chess games. The PGN viewer is the basis for the main screen and the chess board used throughout the program. It’s components for storing chess games, board positions and pieces are also used within the classifier component.

3.3.2 UCI Interface

The UCI Interface used in the program has been taken from the C# library UCI Chess Trainer. It provides all tools necessary for communicating with the chess engine.

3.3.3 Chess Engine

The chess engine used in the program is Spike in the version 1.4. The engine is open source and is considered to be one of the top ten chess engines.
3.3.4 SharpSerializer

The SharpSerializer library is used for serializing the game database after analysis for later use. It transfers an instance of the GameDatabase class into an XML representation and saves it into a file.

3.4 Classifier Architecture

3.4.1 Mistake

The mistake class is the base element of the information stored and processed in the classifier. Every instance of a mistake stores all the information regarding a specific mistake.

This includes:

<table>
<thead>
<tr>
<th>Property</th>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPieceCount</td>
<td>Integer</td>
<td>Number of black pieces on the board.</td>
</tr>
<tr>
<td>Board</td>
<td>2-dimensional array of Pieces</td>
<td>Array representation of the Board.</td>
</tr>
<tr>
<td>Destination</td>
<td>BoardPosition</td>
<td>Destination of the mistake move.</td>
</tr>
<tr>
<td>ECOCode</td>
<td>String</td>
<td>ECO (Chess Opening) Code of the opening played in the game.</td>
</tr>
<tr>
<td>ECOLetter</td>
<td>String</td>
<td>First Letter of ECO Code.</td>
</tr>
<tr>
<td>First2Moves</td>
<td>String</td>
<td>Notation of the first two moves.</td>
</tr>
<tr>
<td>First3Moves</td>
<td>String</td>
<td>Notation of the first three moves.</td>
</tr>
<tr>
<td>FirstMove</td>
<td>String</td>
<td>First Move (Black &amp; White) of the game.</td>
</tr>
<tr>
<td>Game</td>
<td>String</td>
<td>PGN representation of the game.</td>
</tr>
<tr>
<td>Gravity</td>
<td>Integer</td>
<td>Severity of the mistake.</td>
</tr>
<tr>
<td>GravityClass1</td>
<td>Enum GravityClass</td>
<td>Gravity of mistake partitioned into classes. See complex type GravityClass.</td>
</tr>
<tr>
<td>IsEndGame</td>
<td>Float</td>
<td>Probability for mistake to have happened in End game phase.</td>
</tr>
<tr>
<td>IsMiddleGame</td>
<td>Float</td>
<td>Probability for mistake to have happened in Middle game phase.</td>
</tr>
<tr>
<td>IsOpening</td>
<td>Float</td>
<td>Probability for mistake to have happened in Opening phase.</td>
</tr>
<tr>
<td>IsWhiteMove</td>
<td>Boolean</td>
<td>Mistake if White or Black Move.</td>
</tr>
<tr>
<td>Move</td>
<td>Integer</td>
<td>Move count of the move in which the mistake happened.</td>
</tr>
</tbody>
</table>
Analyzing chess game databases

<table>
<thead>
<tr>
<th>MovedPiece</th>
<th>Piece</th>
<th>What kind of chess piece was moved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
<td>String</td>
<td>Official chess notation of the move.</td>
</tr>
<tr>
<td>Quad</td>
<td>Enum Quadrant</td>
<td>In which quadrant of the board did the mistake occur.</td>
</tr>
<tr>
<td>Source</td>
<td>BoardPosition</td>
<td>Where did the move originate.</td>
</tr>
<tr>
<td>WhitePieceCount</td>
<td>Integer</td>
<td>How many white pieces are on the board</td>
</tr>
</tbody>
</table>

Table 2: Mistake Class - Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GravityAsText</td>
<td>String</td>
<td>Returns a textual representation of the gravity class of the mistake. E.g.: “100 – 200”</td>
</tr>
<tr>
<td>QuadrantAsText</td>
<td>String</td>
<td>Returns a textual representation of the quadrant in which the mistake occurred. E.g.: “WhiteKing”</td>
</tr>
<tr>
<td>ToString</td>
<td>String</td>
<td>Returns a textual representation of the mistake. E.g.: “Mistake in move 5 by White: exd5 ”</td>
</tr>
</tbody>
</table>

Table 3: Mistake Class - Methods

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GravityClass</td>
<td>EinBauer, ZweiBauern, DreiBauern, VierBauern, FuenfBauern</td>
<td>Gravity 100-200 centipawns, Gravity 200-300 centipawns, Gravity 300-400 centipawns, Gravity 400-500 centipawns, Gravity 500+ centipawns</td>
</tr>
<tr>
<td>Quadrant</td>
<td>WhiteKing, WhiteQueen, BlackKing, BlackQueen</td>
<td>Kingside of white part of board, Queenside of white part of board, Kingside of black part of board, Queenside of black part of board</td>
</tr>
</tbody>
</table>

Table 4: Mistake Class - Complex Types
3.4.2 Classified Game

The classified game data structure holds all the information concerning a particular game and its analyzed components.

This includes:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzedColorWhite</td>
<td>Boolean</td>
<td>What color does the player have that needs to be analyzed.</td>
</tr>
<tr>
<td>Mistakes</td>
<td>List&lt;Mistake&gt;</td>
<td>List of all mistakes in the game.</td>
</tr>
</tbody>
</table>

Table 5: Classified Game Class - Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzedMove</td>
<td>Void</td>
<td>Save the analyzed move from the engine into the classified game.</td>
</tr>
<tr>
<td>PrintErrors</td>
<td>String</td>
<td>[Deprecated] Returns a String representation of the mistakes.</td>
</tr>
<tr>
<td>PrintMistakes</td>
<td>String</td>
<td>Returns a String representation of the mistakes.</td>
</tr>
<tr>
<td>SaveMistakes</td>
<td>Void</td>
<td>Detects errors within the game by comparing positional information from the engine.</td>
</tr>
<tr>
<td>ToString</td>
<td>String</td>
<td>Returns a textual representation of the analyzed game.</td>
</tr>
<tr>
<td>ToString(int upToMove)</td>
<td>String</td>
<td>Returns a textual representation of the analyzed game up to the move specified in the parameter.</td>
</tr>
</tbody>
</table>

Table 6: Classified Game Class - Methods
3.4.3 Game Database

The game database class holds a set of classified games which have been selected for analysis together by the user. Further, it provides additional information and functionality, such as statistics and mistake clustering.

This includes:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzedPlayerName</td>
<td>String</td>
<td>The name of the player currently analyzed</td>
</tr>
<tr>
<td>BlackGames</td>
<td>Integer</td>
<td>Number black games analyzed</td>
</tr>
<tr>
<td>GameList</td>
<td>List&lt;ClassifiedGame&gt;</td>
<td>A list with all the analyzed games</td>
</tr>
<tr>
<td>MistakeBoard</td>
<td>2-dimensional Integer Array</td>
<td>Counts the mistakes for each square of the board for heatmap representation</td>
</tr>
<tr>
<td>MistakeList</td>
<td>List&lt;Mistake&gt;</td>
<td>A list with all the mistakes that have been found in all the games of the database</td>
</tr>
<tr>
<td>WhiteGames</td>
<td>Integer</td>
<td>Number of white games played</td>
</tr>
</tbody>
</table>

Table 7: Game Database Class - Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>addGame</td>
<td>Void</td>
<td>Adds a new “ClassifiedGame” to the game list</td>
</tr>
<tr>
<td>calculateStatistics</td>
<td>Void</td>
<td>Calculates the Statistics for all features of the mistakes from the MistakeList</td>
</tr>
<tr>
<td>getStatistic(key String)</td>
<td>Statistic</td>
<td>Depending on the value delivered in the key parameter this method returns the Statistic of the feature selected.</td>
</tr>
<tr>
<td>reCalculateStatistics</td>
<td>Void</td>
<td>Recalculate statistics after deserialization.</td>
</tr>
</tbody>
</table>

Table 8: Game Database Class - Methods
### 3.4.4 Statistic

The Statistic Class is a generic container for all features. It uses a string representation of the feature for it to be stored and retrieved. With this mechanism there is no need for different classes for each feature statistic. The class has no public properties and uses the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCount(Object o)</td>
<td>Void</td>
<td>Add 1 to the value for the key provided by the feature (o). This uses the count hashmap.</td>
</tr>
<tr>
<td>addMistake(Object o, Mistake m)</td>
<td>Void</td>
<td>Adds the mistake to the hashmap using the key from the object parameter.</td>
</tr>
<tr>
<td>getTop5Keys</td>
<td>List&lt;Mistake&gt;&gt;</td>
<td>Returns a hashmap with the five entries of the internal hashmap that have the most entries.</td>
</tr>
</tbody>
</table>

Table 9: Statistic Class - Methods
3.1 User Interface

3.1.1 Main Window

Figure 16: Main Window

The main window displays all information needed for starting and overviewing the analysis process. When first started up the user can only select to either load a list of games ("Load PGN" Button) or load a previously analyzed game database ("Load From XML" Button).
Figure 17: Main Window – Analyzing Game

When the user has selected a set of games to be analyzed, the window will display all relevant information about the current game (1). It also displays information on how many games still need to be analyzed (2) and how long this is expected to take (3). Additionally, it displays information on the communication with the chess engine in the text field on the upper right (4) and on the progress of the analysis of the current game (5).
3.1.2 Game Selector

The game selector dialog displays all information on the games that have been found in the selected PGN file. The user can then check whether they selected the correct file and click through the games in the file using the dropdown menu.

![Game Selector](image)

Figure 18: Game Selector
3.1.3 **Statistics Window**

The statistics window is opened after the analysis has been completed. It displays statistical information on the mistakes found in the game database.

<table>
<thead>
<tr>
<th>Number on screenshot</th>
<th>Statistic</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mistake Count</td>
<td>Heatmap representation of the distribution of the mistakes on the board.</td>
</tr>
<tr>
<td>2</td>
<td>ECO Code</td>
<td>Eco Codes of the 5 openings where the most mistakes occurred.</td>
</tr>
<tr>
<td>3</td>
<td>Chess Piece</td>
<td>Top 5 piece types with the most mistakes.</td>
</tr>
<tr>
<td>4</td>
<td>Gravity</td>
<td>Gravity of the mistake divided into 5 classes.</td>
</tr>
<tr>
<td>5</td>
<td>Game Phase</td>
<td>Distribution of the mistakes over the three phases of the game.</td>
</tr>
<tr>
<td>6</td>
<td>Playing Color</td>
<td>Distribution of the mistakes over the colors.</td>
</tr>
<tr>
<td>7</td>
<td>Quadrant</td>
<td>Distribution of the mistakes over the four quadrants.</td>
</tr>
<tr>
<td>8</td>
<td>Overall statistics</td>
<td>General statistics such as games played, mistake count, etc.</td>
</tr>
<tr>
<td>9</td>
<td>Game information</td>
<td>Name of the player analyzed and count of white and black games.</td>
</tr>
</tbody>
</table>

Table 10: Statistics Window
3.1.4 Error Display

The features of each mistake can be displayed using the Error Display window. The user can switch through the games and mistakes within the database and take a look at all the features of each mistake.
4 User Study

4.1 Methodology

Of course the program itself, and the algorithms the program uses, need to be tested and verified. Optimally two groups of chess players would have to be selected one group using the software and the other one training as before. After a reasonable amount of time of one or two years the players strengths, portrayed in their ranking numbers, would be taken and analysis would show if the program and its methods help players play better.

Since the timeframe for this work is 6 months and the timeframe for the user study is even shorter and since the program needed to be finished first, a different approach has been chosen. Tournament chess players, both amateurs and semi-professional players have been selected from a local chess club and an internet chess platform. These players will be given a set of tasks along with a documentation of the program and will document their work with the program in an online questionnaire. The following section will go into detail on what these tasks consist of and what the user has to perform. The documentation of the program can be found in the appendix chapter 6.4 at the end of this work.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Goal of task</th>
</tr>
</thead>
</table>
| Analyze own game | The user needs to select a game of their own or a game of others they know very well and import it into the program. The program will then analyze the game and prompt the user about the mistakes made in the game. | - Test the program in different environments with different data sets  
- Test the analysis capabilities of the program |
| View pre-analyzed data set | A set of XXX games has been pre-analyzed for the test process to speed up. The user will load this data set into the program and then browse and consider its findings and statistics. | - Show statistical and prediction capabilities of the program  
- Let users browse extensive mistake database |
| Free testing     | After that the user may analyze further games and sets of games in order to get more familiar with the program | - Provide further insight into features and improvement possibilities of the program |

Table 11: Tasks for user study

The goal of the user study is to test the program and get feedback from potential users for future improvements.
4.2 Questionnaire

The questionnaire is available in German and English language. This section will contain an explanation of the English version and provide information on the background and goal of the individual questions. The questionnaire is divided into four sections: general questions, usability, game analysis and usefulness of the program. Each section contains a set of questions that need to be answered by the user. The German version of the questionnaire can be found in the appendix chapter 6.5.

General Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>How old are you?</td>
<td>Depending on the number of participants the age of the participants can be used to divide the results in at least two groups: adolescents and adults.</td>
</tr>
<tr>
<td>How long have you been playing chess?</td>
<td>Depending on the number of participants the time a user has been playing chess can be used to divide the results in different groups.</td>
</tr>
<tr>
<td>Do you play chess in a chess club?</td>
<td>Depending on the number of participants the answer can be used to divide the results into two groups.</td>
</tr>
<tr>
<td>What is your player strength (ranking number such as ELO or DWZ)</td>
<td>Depending on the number of participants the results can be divided into two groups: novices and experts.</td>
</tr>
<tr>
<td>How many tournament games do you play in a year.</td>
<td>The answer of this question can be used to determine the potential game database size of the game databases.</td>
</tr>
<tr>
<td>How many of these (above) are recorded?</td>
<td>The answer of this question can be used to determine the potential game database size of the game databases.</td>
</tr>
</tbody>
</table>

Table 12: Questionaire - General Questions
Usability

The next two questions serve to determine whether the program’s usability is good enough to provide easy access to all tools and information needed for the tasks that the user had to complete before the questionnaire.

<table>
<thead>
<tr>
<th>Question</th>
<th>Possible Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>The program...</td>
<td>- ... can be used easily and I did not have to search for needed menu items.  &lt;br&gt; - ... can be used quite easily. For some tasks I had to consult the documentation.  &lt;br&gt; - ... can be used once you have read the documentation.  &lt;br&gt; - ... cannot be used easily even with the documentation.  &lt;br&gt; - ... cannot be used. I did not understand it.</td>
</tr>
<tr>
<td>The program I suitable for fulfilling the tasks of this study.</td>
<td>- I agree completely.  &lt;br&gt; - I agree mostly.  &lt;br&gt; - I agree more or less.  &lt;br&gt; - I mostly do not agree.  &lt;br&gt; - I do not agree at all.</td>
</tr>
</tbody>
</table>

Table 13: Questionaire - Usability Questions
Analysis

This section of the questionnaire provides insight into the algorithms used in the program.

<table>
<thead>
<tr>
<th>Question</th>
<th>Possible Answers</th>
</tr>
</thead>
</table>
| Please state whether the program has found all mistakes in the game you selected. | - The program has found all mistakes.  
- The program has found most of the mistakes   
- The program has found some, but not all of the mistakes.  
- The program only found a few of the mistakes.  
- The program found hardly any or no mistakes.  |
| Please state whether the mistakes the program found really are mistakes. | - All mistakes shown are mistakes.  
- Most of the mistakes shown are mistakes.  
- Around half of the mistakes really are mistakes.  
- Only some of the mistakes really are mistakes.  
- None or hardly any of the mistakes really are mistakes.  |
| The statistics provided by the program are useful. | - I agree completely.  
- I agree mostly.  
- I agree more or less.  
- I mostly do not agree.  
- I do not agree at all.  |

Table 14: Questionaire - Analysis Questions
Usefulness of the program

The last section of the questionnaire serves to provide information about whether or not it makes sense to further develop the program and what features work and which don’t or lack.

<table>
<thead>
<tr>
<th>Question</th>
<th>Possible Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>I think the program is useful.</td>
<td>- I agree completely.</td>
</tr>
<tr>
<td></td>
<td>- I agree mostly.</td>
</tr>
<tr>
<td></td>
<td>- I agree more or less.</td>
</tr>
<tr>
<td></td>
<td>- I mostly do not agree.</td>
</tr>
<tr>
<td></td>
<td>- I do not agree at all.</td>
</tr>
<tr>
<td>Would you use the program?</td>
<td>- Yes</td>
</tr>
<tr>
<td></td>
<td>- No</td>
</tr>
<tr>
<td>If yes, what would you use the program for</td>
<td>- Find my own mistakes.</td>
</tr>
<tr>
<td></td>
<td>- Prepare for the next opponent.</td>
</tr>
<tr>
<td></td>
<td>- For training other players.</td>
</tr>
<tr>
<td></td>
<td>- Other (free text)</td>
</tr>
</tbody>
</table>

Table 15: Questionaire - Usefulness Questions

The last two questions, which both provide a text field for the user to enter their own thoughts, are:

- I liked the following features of the program.
- I did not like or miss the following features of the program.

4.3 Results

As laid out in the beginning of this chapter the goal was to get a set of answers from chess players of different backgrounds and playing strengths. Unfortunately within the three months’ timeframe that was set aside for the gathering of answers only one user filled out the answer form. The user’s answers were favorable for the program but of course cannot be used for any statistical analysis of the program’s usefulness.

However when explaining the program to chess players, the reactions were positive, so that the author still believes in the usefulness and correctness of the program. Unfortunately none of these players, contacted in the early phases of the programming, took the test when it was finished. The further analysis of the prototype will be discussed in chapter 5.
5 Discussion

As proposed in this work, chess programs have indeed long surpassed their creators. Even more so, it has been almost sixteen years since Deep Blue beat Kasparov. Although the detailed description of the strategies used in today’s chess engines are based on the internal structure of Deep Blue, these have not changed dramatically since that time. What has changed dramatically is the computing power available. Deep Blue’s computing power is approximated at around one PetaFLOP [THoCP 2002]. Today’s top-of-the-line Graphics Card, the nVidia Geforce GTX 690 has 5.6 Petaflops. And this is no “super computer”. It can be bought at your local retailer. The hardware around has evolved as well, Solid State Disks’ IO Performance allow for bigger lookup tables for openings, etc.. So it is safe to say, that even mainstream hardware can beat the best chess players.

This is why I suggested a new approach to advancing computer chess programs. Use the computing power not only to beat human players but to teach them how to improve their playing strength. The software presented in this work is a step towards this goal. The user study along with discussions with competitive chess players has shown, that the features provided by the program are very useful for human chess training. The program shows what information can be extracted from chess game databases using data mining techniques and mistake recognition.
5.1 Suggested Improvements to the Prototype

The prototype created for this work is far from perfect. Testing as well as evaluating the user study has shown a great potential for future improvements. A short overview of these improvements is shown in the following table.

<table>
<thead>
<tr>
<th>ID</th>
<th>Type of Improvement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Usability</td>
<td>Make mistake statistics clickable, so that only mistakes with the selected key feature are shown.</td>
</tr>
<tr>
<td>2</td>
<td>Analysis</td>
<td>Fine tune the analysis so that weaker mistakes can be found. The current threshold is one pawn worse evaluation.</td>
</tr>
<tr>
<td>3</td>
<td>Analysis</td>
<td>Provide a feature for different chess engines to be used with the program.</td>
</tr>
<tr>
<td>4</td>
<td>Usability</td>
<td>Let the user chose a destination for the XML-Export.</td>
</tr>
</tbody>
</table>

Table 16: Improvements for prototype
5.2 Future Studies

The program outlined in this thesis is only one part of a bigger framework for adjusting the playing strength of the chess program. Not only randomly or based on a general playing strength, but rather use the player’s own mistakes to improve their play.

![Diagram](chart.png)

Figure 21: Process for adding deliberate mistakes from [Massny 2012]

The player feeds their past game history into the program which is then analyzed as described in chapter 3. The program builds its data as laid out in the algorithms. What needs to be implemented yet are the two marked blocks of the cycle. Using the features from the mistake database, the program will match the current features of the game being played against them and when they match a mistake from the database, it will be planted into the game. The game itself is of course analyzed as well and feeds into the database [Massny 2012].

So my proposition is to use the algorithms and findings from this and feed them into the outlined framework. This could be a way to bring the AI, which has been moving away from human chess players, back to help and better them in their striving for getting better at the “Game of Kings”.

---

48
6 Appendix

6.1 Bibliography


Ertel, Wolfgang; Mast, Florian (2011): Introduction to artificial intelligence. London: Springer (Undergraduate topics in computer science).


Thurau, Christian; Driessens, Kurt; Missura, Olana (Hg.) (2010): Proceedings of the ICML-10 Workshop on Machine Learning and Games. Haifa, Israel.


### 6.2 Table Index

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Chess Piece Values</td>
<td>7</td>
</tr>
<tr>
<td>Table 2</td>
<td>Mistake Class - Properties</td>
<td>32</td>
</tr>
<tr>
<td>Table 3</td>
<td>Mistake Class - Methods</td>
<td>32</td>
</tr>
<tr>
<td>Table 4</td>
<td>Mistake Class - Complex Types</td>
<td>32</td>
</tr>
<tr>
<td>Table 5</td>
<td>Classified Game Class - Properties</td>
<td>33</td>
</tr>
<tr>
<td>Table 6</td>
<td>Classified Game Class - Methods</td>
<td>33</td>
</tr>
<tr>
<td>Table 7</td>
<td>Game Database Class - Properties</td>
<td>34</td>
</tr>
<tr>
<td>Table 8</td>
<td>Game Database Class - Methods</td>
<td>34</td>
</tr>
<tr>
<td>Table 9</td>
<td>Statistic Class - Methods</td>
<td>35</td>
</tr>
<tr>
<td>Table 10</td>
<td>Statistics Window</td>
<td>39</td>
</tr>
<tr>
<td>Table 11</td>
<td>Tasks for user study</td>
<td>41</td>
</tr>
<tr>
<td>Table 12</td>
<td>Questionaire - General Questions</td>
<td>42</td>
</tr>
<tr>
<td>Table 13</td>
<td>Questionaire - Usability Questions</td>
<td>43</td>
</tr>
<tr>
<td>Table 14</td>
<td>Questionaire - Analysis Questions</td>
<td>44</td>
</tr>
<tr>
<td>Table 15</td>
<td>Questionaire - Usefulness Questions</td>
<td>45</td>
</tr>
<tr>
<td>Table 16</td>
<td>Improvements for prototype</td>
<td>47</td>
</tr>
<tr>
<td>Table 17</td>
<td>Answers Questionaire</td>
<td>62</td>
</tr>
</tbody>
</table>
6.3 Figure Index

Figure 1: Chess Board - Initial Position ................................................................. 5
Figure 2: Chess Piece Movements ........................................................................ 6
Figure 3: Check Mate Positions ............................................................................. 6
Figure 4: Alpha Beta Pruning Algorithm [Knut, Moore 1975] .............................. 11
Figure 5: Alpha-Beta-Search tree after 4 half-moves [from Ertel, Mast 2011] ....... 11
Figure 6: Dimensions of chess knowledge [Gobet, Jansen 2005] ......................... 14
Figure 7: Complete Algorithm ............................................................................. 20
Figure 8: Algorithm - Parse File ......................................................................... 21
Figure 9: Algorithm - Analyze Game ................................................................. 22
Figure 10: Algorithm - Find Mistakes ................................................................. 23
Figure 11: Algorithm - Extract Features .............................................................. 25
Figure 12: Algorithm - Extract Features - Opening ............................................ 26
Figure 13: Algorithm - Extract Features - Middle game ..................................... 27
Figure 14: Algorithm - Extract Features - Endgame .......................................... 28
Figure 15: Algorithm - Statistics ........................................................................ 29
Figure 16: Main Window .................................................................................... 36
Figure 17: Main Window – Analyzing Game ....................................................... 37
Figure 18: Game Selector .................................................................................... 38
Figure 19: Statistics Window .............................................................................. 39
Figure 20: Error Display .................................................................................... 40
Figure 21: Process for adding deliberate mistakes from [Massny 2012] ............ 48
6.4 User Study Tasks

The following section displays the tasks that were needed for the evaluation in the user study.

CHESS TRAINING – TASKS

Task 1 – Install the program

- The program only runs under Microsoft Windows
- Unzip the files into a folder
- Run “ErrorClassifier.exe”

Task 2 – Analyze a game

Pick a game you know well, one of your own or a game you recently analyzed.

Open the program. You should see the start screen.

Click on load pgn.
Pick a pgn file.

Click on select. If the file contains multiple games, they all will be analyzed.
Click “Start Analysis”. This might take a while. The time remaining will be displayed at the bottom. Best you get a coffee.

Take a look at the statistics. They are not that useful if you only analyze one game.
Close statistics window, click on “Show Errors”. Take a look at the mistakes the program found.

**Task 3 – Look at a game Database**

Open the program, click on “Load from XML”

Take a look at the statistics and mistakes. => You are finished and can start to answer the questionnaire.
6.5 Fragebogen – Deutsche Übersetzung

Schachtraining durch statistische Fehleranalyse

Hallo liebe Teilnehmer,

herzlichen Dank, dass Sie sich die Zeit nehmen, das Ihnen zur Verfügung gestellte Programm zu bewerten. Als Dankeschön für Ihre Zeit werden unter allen Teilnehmern Amazon Gutscheine verlost. Dafür möchte Ich Sie bitten, auch Ihre Email-Adresse zu hinterlegen. Die Email-Adresse wird nur für die Verlosung herangezogen und nicht weiter gespeichert oder für die Analyse der Daten verwendet. Alle von Ihnen eingegebenen Daten werden anonymisiert und vertraulich behandelt. Bevor Sie mit der Beantwortung der Fragen beginnen, möchte Ich Sie bitten, die Anweisungen und Hilfestellungen für den Programmtest zu beachten, die Sie unter folgendem Link finden:

LINK HIER

Bei Fragen stehe ich Ihnen gerne unter c.massny(at)gmx.de zur Verfügung.

Ich wünsche Ihnen viel Spaß mit dem Programm und bei der Beantwortung der Fragen.

Mit freundlichen Grüßen,
Christian Massny

* Required

Allgemeine Angaben

Zunächst möchte Ich Sie bitten, ein paar allgemeine Angaben zu machen.

Wie alt sind Sie? *

Bitte geben Sie hier Ihr Alter ein.

Wie lange spielen Sie schon Schach?

Bitte geben Sie hier an, wie lange sie bereits Schach spielen.

Spielen Sie im Verein Schach? *

Bitte geben Sie an, ob Sie an Vereinsturnieren, Punktspielen oder ähnlichen teilnehmen.

Welche Spielstärke haben Sie? *

Bitte geben Sie hier, wenn vorhanden, Ihre DWZ/ELO an. Wenn Sie keine DWZ haben, oder nicht wissen, worauf die Frage abzielt, geben Sie bitte 0 ein.
Wie viele Turnierpartien spielen Sie im Jahr? *

Bitte geben Sie hier an, wie viele Partien Sie im Jahr auf verschiedenen Turnieren spielen. (Regionalmeisterschaften, Open, Vereinsmeisterschaften, Schnellschachturnier, etc.)

Wie viele davon werden aufgezeichnet? *

Bitte geben Sie an, wie viele der oben genannten Partien mitgeschrieben und später veröffentlicht werden.

Bitte geben Sie Ihre Email-Adresse an.

Benutzung des Programms

Im nächsten Abschnitt werden Sie gebeten, Angaben über die Benutzerführung und -freundlichkeit des Programms zu machen.

Die Benutzung des Programms... *

- hat sich mir direkt erschlossen. Ich habe nicht suchen müssen, um benötigte Menüpunkte zu finden.
- erschließt sich leicht, bei einigen Optionen musste ich jedoch die Dokumentation zu Rate ziehen
- lässt sich mit Hilfe der Dokumentation erschließen
- erschließt sich selbst mit der Dokumentation nur schwer.
- habe ich nicht verstanden.

Mit dem Programm lassen sich die Aufgaben aus der Fragestellung erfüllen. *

- Stimme voll und ganz zu.
- Stimmer eher zu.
- Stimme teils zu.
- Stimme eher nicht zu.
- Stimme gar nicht zu.
Analyse der Partien

Im nächsten Abschnitt werden Sie gebeten, die Qualität der Analyse zu bewerten und Angaben zu Ihrer Einschätzung der Nützlichkeit des Programms zu machen.

Bitte geben Sie an, ob das Programm die Fehler der von Ihnen gewählten Partie gefunden hat. *

Sie wurden gebeten eine Ihrer oder eine Ihnen bekannte Partie vom Programm analysieren zu lassen. Bitte geben Sie hier an, ob das Programm alle Ihnen bekannten Fehler gefunden hat.

- Das Programm hat alle Fehler gefunden.
- Das Programm hat die meisten Fehler gefunden.
- Das Programm hat einige, jedoch nicht alle Fehler gefunden.
- Das Programm hat nur wenige der vorhandenen Fehler gefunden.
- Das Programm hat keine oder nahezu keine Fehler gefunden.

Bitte geben Sie an, ob die vom Programm gefundenen Fehler auch tatsächlich Spielfehler darstellen. *

Sie wurden gebeten eine Ihrer oder eine Ihnen bekannte Partie vom Programm analysieren zu lassen. Bitte geben Sie hier an, ob die vom Programm gefundenen Fehler auch tatsächlich Fehler waren.

- Alle gezeigten Fehler sind auch Fehler.
- Die meisten der gezeigten Fehler sind Fehler.
- Gut die Hälfte der Fehler sind auch tatsächlich Fehler.
- Nur einige der Fehler sind tatsächliche Fehler
- Keine oder nahezu keine der gefundenen Fehler sind tatsächlich Fehler.

Die vom Programm dargestellten Statistiken bieten wertvolle Informationen. *

Bitte geben Sie an, ob die Statistiken, die Sie bei der Ihnen zur Verfügung gestellten Partiensammlung angeboten bekommen, Informationen vorhanden sind, die Sie verwenden können.

- Stimme voll und ganz zu.
- Stimme eher zu.
- Teils, teils.
- Stimme eher nicht zu.
- Stimme gar nicht zu.
Nützlichkeit des Programms

Im nächsten Abschnitt werden Sie gebeten, einzuschätzen, für wie nützlich Sie das Programm erachten.

Ich finde das Programm nützlich. *

Bitte geben Sie hier an, ob Sie der Aussage zustimmen.

- Stimme voll und ganz zu.
- Stimme eher zu.
- Teils, teils.
- Stimme eher nicht zu
- Stimme gar nicht zu.

Würden Sie das Programm benutzen? *

Bitte geben Sie hier an, ob Sie das Programm als Spieler oder Trainer benutzen würden.

- Ja
- Nein

Wenn Sie die obige Frage mit Ja beantwortet haben, wofür würden Sie das Programm benutzen?

- Eigene Fehler finden
- Vorbereitung auf den nächsten Gegner
- Zum Training anderer Spieler
- Other:

Folgende Dinge, Funktionen haben mir am Programm gefallen. *

Folgende Dinge, Funktionen haben mir am Programm NICHT gefallen.
### 6.6 Questionaire – Answers

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>How old are you?</td>
<td>58</td>
</tr>
<tr>
<td>How long have you been playing chess</td>
<td>30 years</td>
</tr>
<tr>
<td>Do you play in a chess club?</td>
<td>Yes</td>
</tr>
<tr>
<td>What is your playing strength.</td>
<td>1750</td>
</tr>
<tr>
<td>How many tournament matches do you play per year?</td>
<td>3</td>
</tr>
<tr>
<td>How many of those are written down?</td>
<td>3</td>
</tr>
<tr>
<td>Please state your Gameknot user name.</td>
<td>mlazar</td>
</tr>
<tr>
<td>The program...</td>
<td>... can be used quite easily. For some tasks I had to consult the documentation.</td>
</tr>
<tr>
<td>The program I suitable for fulfilling the tasks of this study.</td>
<td>I mostly agree</td>
</tr>
<tr>
<td>Please state whether the program has found all mistakes in the game you selected.</td>
<td>The program has found most of the mistakes</td>
</tr>
<tr>
<td>Please state whether the mistakes the program found really are mistakes.</td>
<td>Most of the mistakes shown are mistakes.</td>
</tr>
<tr>
<td>The statistics provided by the program are useful.</td>
<td>I mostly agree.</td>
</tr>
<tr>
<td>I think the program is useful.</td>
<td>I mostly agree</td>
</tr>
<tr>
<td>Would you use the program, when it’s complete?</td>
<td>Yes</td>
</tr>
<tr>
<td>If yes, what would you use the program for</td>
<td>Find my own mistakes</td>
</tr>
<tr>
<td>I liked the following things about the program.</td>
<td>nice format</td>
</tr>
<tr>
<td>I did NOT like the following things about the program</td>
<td>was a bit leery of download</td>
</tr>
</tbody>
</table>

Table 17: Answers Questionaire