Preface

This is a Master Thesis in Applied Computer Science carried out by Thomas Mellemseter during the spring semester of 2017 at NTNU Gjøvik. The idea of researching Darkchess was brought up in a conversation with Associate Professor Simon McCallum, where the general topic was about the relevance in researching artificial intelligence within traditional chess.

The reader should have a good understanding of probability theory as well as about computer programming. The more chess knowledge the better, but should not be necessary. However, knowing the basic rules and how to play traditional chess is assumed.

May 31, 2017

Acknowledgment

First and foremost I would like to thank my supervisor Associate Professor Sule Yildirim-Yayilgan of the Department of Information Security and Communication Technology at NTNU Gjøvik. She was always available and I am grateful for the help and the feedback she provided during this semester. This also applies to my co-supervisor Professor Rune Hjelsvold of the Department of Computer Science at NTNU Gjøvik. He always provided with helpful feedback that motivated the work throughout this semester. Thank you both.

A special thanks go to Grandmaster (GM) John Ludvig Hammer for finding time and answering questions regarding darkchess.

Finally, I must express my gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you.

Abstract

Two player strategy games with partially observable environments face challenges regarding reasoning under uncertainty. The aim of this thesis is to investigate darkchess with focus on searching and evaluating actions based on partial knowledge. This has been approached by risk assessing threats within the unobservable part of the environment. A working darkchess agent has been developed, where multiple tests between different agents has been conducted, as well as a user test. The results, based on statistical analysis, indicate that a modified alpha-beta search algorithm with risk assessment and a simplified evaluation function approach semi-decent playing strength.

Contents

List of Figures

List of Tables

1 Introduction

Computers have become a vital tool in everyday life. They are installed in cars, used for communication, and are placed in shops guiding business intelligence, e-commerce, and enterprise asset management. Computers are great in calculating and processing a large amount of data, often used to support people who face difficult decision problems. They make it easier and faster to decide, and more effectively, especially in situations which entail uncertainty.

1.1 Reasoning under uncertainty

Making decisions based on incomplete information happens often on a daily basis. Teachers do not know exactly what each student understands, while doctors do not know exactly what is wrong with a patient. Diagnosis, for instance, involves uncertainty where it is unrealistic in knowing all possible factors that add up the symptoms. This is known as the qualification problem, meaning one can never know the exact outcome of an action in general. This is why reasoning under uncertainty is important, as one can draw a conclusion of what is most common, or more likely to happen based on previous knowledge and experience.

The best one can do is provide a degree of belief in these situations, in which probability theory comes in handy. With it, one can outline the certainty (or uncertainty) of outcomes and thereby the qualification problem diminishes. From the perspective of a computer agent, uncertainty boils down to not knowing which state it is currently in, or which state it might end up in after executing an action. [1].

The qualification problem is a common feature where reasoning under uncertainty is needed, and it is not only typical under the medical domain but applies to other judgmental domains as well. This includes law, business, design, automobile repair, gardening, dating, and not to forget computer games. [2, 1, 3].

1.2 Keywords

- Probability Theory
- Artificial Intelligence (AI)
- Chess AI
- Partial observable environment
- Imperfect-information games
- Decision making

- Representing uncertainty
- Reasoning under uncertainty

1.3 Problem Description

All factors that can help a person decide on an action, whatever action it might be, are rarely known in advance. This results in actions being based on what is already known and assumptions made on uncertain information. Making decisions based on partial knowledge increases the complexity of AI systems, because of the added uncertainty. Knowing if a move is good or bad in darkchess is a non-trivial task, more so, compared to traditional chess as it is a perfect information game.

Darkchess has an environment that is static, sequential, deterministic and partially observable. A static environment means that nothing changes while the agent decides what to do, in other words, since chess is turned based, the state will only change based on the agent's action. It is sequential since past actions determine which actions become available as the game progress, and deterministic as a specific move in a specific state will always guarantee the same change of the board-state. However, the entire board is only partially observable, meaning a player does not see every piece on the board. The partial observability is also dynamic, meaning available actions may affect the degree of observability.

As presented in Chapter 3, darkchess is a new research domain related to AI and reasoning under uncertainty. Understanding more about the uncertainty of darkchess will reveal challenges and possible solutions that can be generalized to other similar complex domains and environments. More specifically aimed at 2 player strategy games.

The research boils down to reasoning about outcomes based on uncertain information. For darkchess it means finding moves that approach the agent's goal where the outcome does not entail high risk for the opposite to happen, moving away from the goal. That being said, the uncertainty is about being unaware of opponent's moves. This, and darkchess in general, is further explained in Chapter 2 demonstrating the severity of not knowing all the information.

1.4 Research Questions

1. What design choices have the most impact on the AI decision making?
2. To what degree does the uncertain information affect the decision maker?
3. How stable is the applied evaluation function over the duration of a game?
4. How important is deep move searching in darkchess?
5. How does uncertainty affect traditional chess evaluation methods?

The first research question above covers explicitly how the AI agent has been developed, which is at the core of the following research questions. More specifically, it will answer what techniques have the most impact on how the agent searches and distinguishes between different moves. The second question focuses on how much the uncertainty aspect will affect whether the agent chooses goal-approaching moves or not, where the third address how stable the applied technique is over the duration of a game. The fourth research question focuses on the impact for searching deeper for improving actions, and what effects it might have. This is of course highly dependent on the design choices for the search algorithm. The last question covers how the uncertainty affects traditional techniques for evaluating the state.

Darkchess is basically traditional chess with applied uncertainty. Overall, these research questions will cover how uncertainty affects the traditional chess domain indirectly, but more important address the challenges and possible solutions to handle searching for actions within domains similar to darkchess.

2 Darkchess

2.1 Origin

Darkchess was invented in 1989 by Jens Bæk Nielsen and Torben Osted. The variation is inspired by another chess variant called Kriegspiel chess where the goal is to simulate real war. In Kriegspiel, neither player will never see any of the opponent's pieces during a game. When an illegal move is attempted, feedback is provided by a referee only to the player to move. The referee is a necessary third player which has complete knowledge of the game which can communicate to one or both players. In darkchess, a player has limited information about the placement of the opponent's pieces. In other words, one is unable to see the entire board, only their own pieces and the squares these pieces can legally move to. This is the main difference compared up against Kriegspiel, which includes showing opponents pieces if the player are able to capture it. This can be seen in Figure 1. It is called darkchess because it feels like moving in the dark. Both Osted and Nielsen played a correspondence game that lasted for little under 2 years and 7 months from 1989 to 1992. The entire game is available online with comments from blacks perspective [4]. They discarded the en-passant move rule as they claim it would be difficult to handle and would not influence the game anyhow. This is a special pawn rule; normally a pawn can only capture pieces that are one square diagonally in front of it. However, there is an exception, which involves another special pawn rule, the rule where pawns can move 2 squares forward from their initial position. Capturing en-passant is available when a player moves a pawn 2 squares while it passes an opponent pawn in either neighboring files. The opponent can in the next move, and only in the next move, capture the *passing* pawn just as if it moved 1 square. [4]

Figure 1: Illustrating observable and unobservable squares

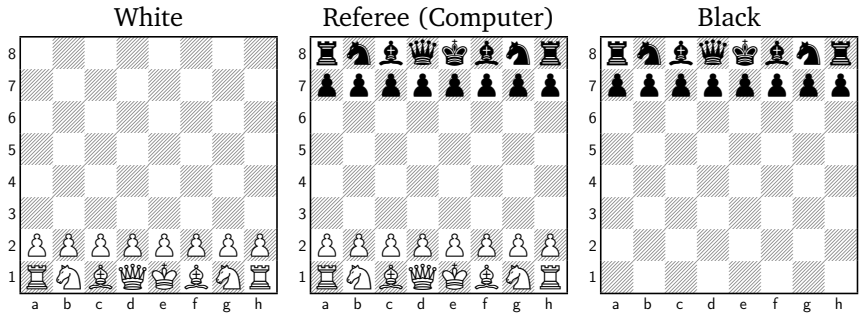


Figure 2: Initial board state for each player

2.2 Rules

The rules of darkchess are mostly the same as traditional chess. The same pieces are used and it is played on an 8x8 chess board. However, the goal is not to checkmate the opponent's king, but to capture it. Castling, for instance, is possible both into and out of check, unlike traditional chess. This means the king can also move through an attacked square when castling. As mentioned, the special en-passant rule is omitted, although it is stated of being allowed according to wikipedia. Here, it has been discarded, as it would only affect a small portion of states when playing, agreeing that it would not influence most games in any decisive matter. To simplify the game, squares directly in front of pawns is made observable, even though it cannot attack it. This includes revealing pieces that are positioned two squares away from pawns in their initial position. This conforms with the original idea of the game but deviate from what is stated on wikipedia.

By preventing a player of knowing the entire board-state while playing, the environment is partially observable. As traditional chess is a perfect-information game, dark chess becomes an imperfect-information game. This means that each player has their own board when playing, and in order to make this work, a third player needs to be responsible for what each player can see, acting as a referee. Form the correspondence game between the founders, the acting referee was Nielsens wife. What each player see is illustrated in Figure 2. Unobservable squares in the developed application are displayed with an X, also referred to as dark squares. This is implemented to prevent confusing dark squares with observable but empty squares. This is shown in Figure 1, which also show that a4 is visible due to the pawn on a2. Because of this rule, it is possible for a player to make a move that is unobservable for the other player (a dark move). This way, the referee needs to inform each player when it is their turn to move. [4, 5]

2.3 Strategy

Just as in traditional chess the white player start by making a move. Before any move has been made, both player has complete knowledge of all pieces on the board as the initial piece position is the same for both players. This makes whites first move, whatever move it might be, involve no risk of being captured. Playing as black, on the other hand, 5 of the 20 first moves involve a risk of losing a piece. This feature grants white with a big advantage as it becomes easier to start taking control over the center of the board. This is just as important in darkchess as in traditional chess. Grandmaster (GM) John Ludvig Hammer and Norway's second best chess player (as of April 2017) [6], answered and discussed the game of darkchess over a Skype conversation. He was unfamiliar with the Darkchess variation, but knew about Kriegspiel, along having played numerous other chess variations. He claimed white had a very big advantage as it is the first player to move. He also pointed out that the game is not too much about taking chances, but about playing more positional chess, which is to place pieces on good squares helping in taking control over the board. By gradually gaining more space on the board results in more move options which in turn increases the opportunity for an attack.

In darkchess, there exist 4 different kinds of moves related to how much information the other player can infer. The first is moving a piece which is visible to the opponent and placing it on another visible square. These moves are simply referred to as *known moves*. Another move, the exact opposite, is where the opponent does not see either what piece is being moved or where it was placed. These are referred to as *dark moves*, and one can only reason for what move happened. The other 2 move types reveal some information, reducing the number of possible moves that might have happened. The easiest of the two is seeing what piece is being moved, but not knowing where it was placed. The other is not knowing where or what exact piece was picked, but knowing where it was placed. The latter makes it more difficult to reason as the number of possible moves increases with the number of possible pieces it could be. For instance, discovering a knight in a middle game, it becomes more challenging knowing which knight it could be compared to seeing the knight disappear. These moves are referred to as *retreat moves* and *surprise*

From\To	**Visible**	**Dark**
Visible	Known Moves	Retreat Moves
Dark	Surprise Moves	Dark Moves

Table 1: The 4 different move types in Darkchess

moves respectfully. These main move types are displayed in Table 1

In addition does it exists 2 special move types in darkchess which both relate to losing an unprotected piece. The first special move is when the attacking piece is known. All squares that are revealed by the unprotected piece becomes dark squares, while the known attacking piece seems to have disappeared. However, since only one move can occur at a time, this means, the known opponent piece had to capture the unprotected piece making this in practice a known move. The second special move is when the attacking piece comes from the dark space, thus being unknown. In these scenarios, the unprotected piece only gets removed from the board, and the player can only reason for which piece the opponent attacked with. Both of these special moves are displayed in Figure 3. The latter is referred to as *adverse surprise moves*.

(a) Unprotected pawn on e5

(b) The knight captured on e5.

(c) Unprotected pawn on e5

(d) No knowledge about the attacking piece

Figure 3: Darkchess special moves: (a) and (b) shows a semi-retreat move, which in practise become a known move, (c) and (d) show a similar scenario, only which piece captured the pawn on e5 is unknown.

Hammer believes that material values are the same as in traditional chess, meaning queens are more valuable than rooks, rooks more valuable than bishop & knight which again are more valuable than pawns. On the other hand, Nielsen question having a material advantage being a decisive factor as in traditional chess. This is stated in his analysis of the correspondence game online where he plays as black, specifically on move 9 where he loses material. Nielsen's chess knowledge is unknown, but Hammer pointed out that blacks opening was horrible in this correspondence game.

2.4 Uncertainty

The uncertainty in darkchess changes the concept of tactics compared to traditional chess, such as pinning a piece for instance. In traditional chess, a pinned piece is a piece that cannot move because otherwise the king would be exposed to an attack. However, in dark chess there also exists dark pins, making a player unaware of such scenarios. By moving the pinned piece means putting their own king open to being captured, ending in a loss. This is demonstrated in Figure 4 which illustrates the severity of only knowing part of the environment, and the uncertainty that follows.

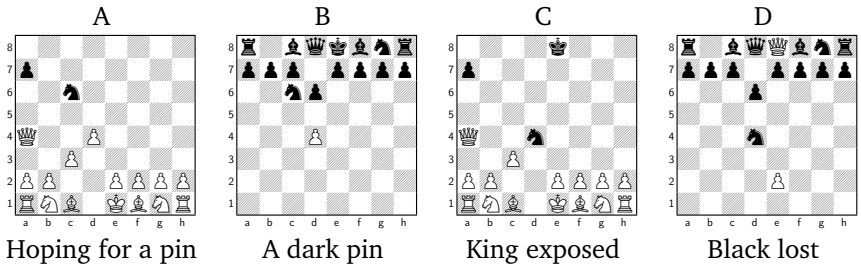


Figure 4: Showing the idea behind a dark pin

In Figure 4 A, White has moved *1.c3 ? 2.d4 ? 3. Qa4* where a question mark indicate a dark move by black. After white's last move *Qa4* reveals that one of the dark moves is *Nc6*, and black has not moved their *a* pawn. However, white does not know if the knight is in a pin since no information is given whether black has moved their *d* pawn, or more specifically d6. Since white's second move was *d4* they know black has not played *d5*. The position of black's king is known in this position since both players start with the same piece placement and the fact that the quickest sequence in order to move the king is by first moving one of the pawns in front of it followed by the king. This has not happened since white know of the Nc6 move.

Figure 4 B show the same position as in A but from blacks perspective. Since black is unaware of the queen on a4, and black has in fact moved *d6* the knight is in

a dark pin. Black does not consider this possibility and instead risks the chance that white is not protecting the pawn. This is obviously a huge blunder, ending black to lose the game, but keep in mind this continuation is to demonstrate the severity of these types of positions. Thinking the pawn on *d4* is unprotected means white had moved their queen away from the *d* file. This requires another pawn move, either *c* or *e*, and in order to prevent them to protect on d4 it has to be specifically *c4* or *e4*. Black has no information of this. Although black is right that the white's queen has moved, unfortunately, it continues to protect on *d4*. That being said, it is obvious why taking on *d4* is a blunder. In C, white is happy to see blacks king in a position where it is white to move, making it fairly easy to win the game. This is shown in D from blacks perspective.

2.5 Game complexity

Measuring game complexity gives an insight of its state-space size and game-tree size. The former refers to the amount of different states allowed by the game rules while the latter refers to a number of different games that can be played. It is well known that solving chess by brute force is highly impractical. [7] Considering all permutations of all 32 pieces being able to be placed on all 64 squares add up to ${}_{64}P_{32} \approx 10^{54}$ possible states [8]. However, this includes states with illegal piece placement such as white pawns on the first rank and both kings in check. It would also be wrong stating this number being an upper bound as it does not consider pawns being promoted. A lower bound of the state-space complexity was calculated in 1950 by Claude Shannon of being roughly 10^{43}. He also estimated the game-tree complexity to be 10^{120}. Since then, these numbers have been re-estimated to 10^{47} and 10^{123} respectfully. In other words, the decision problem in chess grows exponential, meaning chess is EXPTIME classified based on computational complexity theory. [7, 9, 10]

Considering darkchess from the referees perspective, one might think these numbers would be accurate. However, the average number of legal moves in darkchess is larger since one are able to move in and out of check. Nevertheless, these numbers do not make much sense from either player's perspective. Their view of the complexity is much higher when considering their belief state. To simplify, lets round the space complexity to an exponent of 50 instead of 43. Then imagine all piece placements for white being 10^{25} where each has on average 10^{25} positions of black pieces. This means the number of unique belief states is more closer to $10^{25} \times 2^{10^{25}}$. These calculations were done for the similar chess variation Kriegspiel. In this variation, one is unable to observe any of opponent's pieces at all and has to rely on a referee knowing if a move is legal or not. That being said, as darkchess are able to observe a portion of the opponent's pieces, limiting the belief-state-space

complexity. However, how much a player are able to observe of the opponent highly depends on both players piece placement and would, therefore, vary in each game. This makes the belief-state complexity being dynamic. Having an option making people play against each other could collect data about the average belief-state for each game which is again averaged across many games, getting a closer approximation. This also concerns the number moves that is executed in the average darkchess game. For Kriegspiel, this has been calculated of being 52. Unfortunately, this is outside of the scope of this thesis. However, one observation worth mentioning is that the belief state complexity of darkchess can not be any worse than that in Kriegspiel, which has an upper bound of the belief state not exceeding 10^{130}. [8]

3 Related Work

Searching the Internet for *Dark Chess* will provide with results referred to as *Banqi*, also known as *Chinese dark chess*, or even *Half chess* as it only uses half of a chess board. This is a very different variation than what is referred in this paper. However, both games share the uncertainty element which makes this field slightly relevant.

No academic research projects have been found that looks into darkchess. This indicates a new microcosm for uncertainty with AI, although there exist similar research projects in the area [11, 12, 13]. The closest chess variant seems to be *Kriegspiel chess* where players never see any of their opponent's pieces during a game. This variation has been mentioned in Chapter 2. One main difference is that the referee in Kriegspiel also has to give feedback to each player if an illegal move is attempted. In such scenarios, it is a good guess that there is an opponent piece blocking its path. However, as soon as a move is legal, it is played right away, meaning a player needs to commit to any move. In other words, it is considered risky wishing for illegal moves in order to gain more knowledge of the environment. This also makes the information asynchronous, meaning a player does not know what information the other player has obtained. This element is exactly what darkchess prevents by revealing the squares the pieces can move to and/or attack.

3.1 Kriegspiel Chess

A Ph.D conducted by Favini in 2010 [8] has been looking at Kriegspiel focusing on abstracting the belief state, a method referred to the use of *metapositions*. The goal is to transform Kriegspiel over to a perfect-information game such that minimax search algorithm can be applied. With metapositions, the problem size decrease, which is achieved by merging multiple game states into a data structure which is more manageable. What kind of states to merge depends on the moves that are possible from the current state. In other words, the state after performing similar moves gets merged to a single state. The main goal of the approach is to provide players with the illusion of perfect information, making it possible to utilize an evaluation function. Because of the partial observable environment in Kriegspiel, states which might not be compatible with a game history could be merged, making metapositions able to represent states in a very compact form. This is claimed to make it easier to develop evaluation functions for the entire metaposition. This concept also introduces *pseudolegal moves*, and *pseudo pieces*, since metapositions can contain unreachable states, these will also contain illegal moves. Pseudo pieces

can duplicate themselves and also be placed on squares which are already occupied by another pseudo piece. These move only on the opponent's turn, imagining that each piece gets duplicated and moved to a legal square based on the feedback from the referee. For each move by a pseudo piece their uncertainty increases, but by moving own pieces over squares which have one or more pseudo piece gets removed, and their uncertainty decrease. Each square on the board is also linked with an aging value. This is increased as the game progress and no inference happens on consecutive moves for every square. This is used to encourage visiting aged squares. [8]

One aspect of the game, makes it asymmetrical, meaning a player does not know what information the other player has gained from the referee, such as illegal move attempts. Extracting such information is therefore deemed extremely important.

Favini look into improving strategies in the end game, but also compare different MCTS approaches to Kriegspiel. The game is noted to be a difficult game for computers to master, although the gap between human strength is reduced, with a developed program being among the top 20 players over at the Internet Chess Club. [8]

3.2 Monte Carlo Tree Search

Favini acknowledges MCTS for being appropriate in environments which struggle with the use of a minimax approach, specifically mentioning Go. Here MCTS is described being merely a framework for which one can experiment with different strategies for each step that is MCTS. These are selection, expansion, simulation, and backpropagation. Selection finds the most visited leaf node of a game tree to expand further before simulating the rest of a game. In the end, it propagates its result back to the root node. These steps are performed multiple times, where a move is selected based on the number of visits a node gets. 3 different MCTS methods have been compared, one being based on previous research using random sampling. The other two methods is very similar, only simulating the referees messages, avoiding randomness which was a factor for performing inaccurate moves. Simulating full games in the first approach were *too detrimental* making the other approaches only simulate a portion of a game, determined by the number of moves to be played. This is also the difference between the last approach, which only simulates games by one move. However, this includes quiesce search, and the smaller the game length the more simulations are generated.

This first method failed, with a win ratio below 2%, barely winning over a random agent. This reason was mainly due to depending on randomness itself. The last approach performed best, which is argued because of basically being UCT selection. Overall, Favini conclude that an MCTS algorithm can do good within a

reasonable amount of time in a domain like Kriegspiel.

Research conducted by Yen, Chou, Chen, Wu, and Kao look into how MCTS can be applied to Chinese Dark Chess (CDC) programs. Most CDC programs use alpha-beta search with chance nodes to handle the uncertainty. This results in a massive increase in the branching factor, even bigger than Go, making alpha-beta unable to search deeply. The use of MCTS with chance nodes is researched with random nodes for dealing with a partially observable environment. Variations of this approach has been shown possible for other games such as Backgammon, Kriegspiel, and Poker. [11]

One important difference between CDC and darkchess are that in the Chinese variation both players see the same board where all pieces are shaped as a flat cylinder only showing the type of the piece on one side. In the opening, all pieces are faced down and randomly placed on the board. The game starts when a player flips a piece, revealing its color and type. This is the uncertainty of the game, where players can flip pieces that belong to the opponent. To handle this during a search, the MinMax game tree structure has been modified adding chance nodes in order to deal with flipping moves. The child nodes of a flipping move represent the likelihood of which piece it might be.

The paper contributes with successfully implementing a nondeterministic MCTS (NMCTS) to a Chinese Dark Chess program named *DIABLO*. It won 4 gold medals in computer CDC tournaments as well as a silver medal in Computer Olympiad in 2013. [11]

Another research by Jouandeau and Cazenave in this field experiment on both group- and chance-nodes, where group-nodes consider all revealing moves from a given position. This is to reduce the branching factor created by flipping moves. Other than that, they show various experiments to reveal the most promising polices. The four basic policies are *Random*, *Capture*, *Avoid*, and *Trap* which corresponds to their respective goal. For instance, *Avoid* means avoiding opponents capture, while *Trap* means minimizing opponents moves. Their experiments show that some polices are slow, namely Avoid and Trap. Further, only the best policies are picked checking their ability to reduce the number of drawn endgames [12]. This part is not as relevant since draw rules for CDC are slightly different from darkchess.

3.3 Alpha Beta Search Alternatives

There has been numerous research on game tree search, where Junghanns provided in 1998 a survey of the field [14]. The focus is on zero-sum 2 player games, where alpha-beta pruning has shown to be a successful search method, especially in perfect-information games. The survey acknowledges the limitations of alpha beta

pruning and describes and assesses alternative game tree search methods which do not share the same problems. When searching for solutions, it is ideal to reach the leaf nodes which contain the game theoretic values, meaning typically win, loss, or possible draw. For games with huge branching factors, it is impractical to search down to the leaf nodes as it would be too time-consuming, or require an unreasonable amount of memory while searching. That being said, alpha-beta pruning faces a depth limit which again losses the game theoretic value as its gets replaced by an estimate calculated by an evaluation function. This, in turn, raises other issues as well. It should also be mentioned that alpha beta is a significant improvement to minimax search as it eliminates searching big portions of a game tree, making it plausible to search even deeper. Nonetheless, the most notable are these issues: scalar value, stopping, expand next, and opponent. As evaluation functions compress the knowledge to a single scalar value, it could potentially lose information that could be useful during a search. In cases where the search suggests one of the many successor nodes, the best is the most appropriate to select which is determined by the evaluation function. The expand next issue is about the fact that alpha-beta pruning follows a depth-first search strategy, which depends on the order that the successors expand next. The last issue, opponent, is about the agent assuming the opponent is using the same heuristics. With different heuristics imply that one agent is better, and when an agent thinks the opponent evaluates the position equally makes it hopeless for the weaker agent. [14, 15, 16]

The are also alpha beta enhancements which partially solve some of the issues. Iterative deepening for instance, in practice, addresses the stopping issue by instead of setting a fixed depth, increasing the depth by one until a time limit has reached. According to the chess-programming wiki, it is the basic time management approach for Depth First Search (DFS) algorithms. It allows for storing information from earlier iterations which can be used in later iterations. However, this is not claimed to be an optimal solution. Move ordering is another enhancement which potentially can increase the number of cut-offs, focusing on good moves. How to sort moves, could be used by information obtained by previous iterations, which addresses the issue of *expand next*. It also claims that capturing moves are usually good, as well as, good moves in sibling positions could be prioritized, given they are legal. [14, 17, 16]

The notable alpha-beta alternatives discussed are a product-propagation procedure, B*, Bayesian game tree search, speculative play, and opponent modeling. The general idea of the product-propagation procedure is that the evaluation function does not return either a game theoretic value or a scalar value, but instead return a probability of the position to be a forced win. This is calculated by multiplying the probability of child nodes having a forced win. Unfortunately, this assumes sib-

ling nodes are independent which is claimed to be unrealistic in general. B* is a best-first search, meaning it finds the best child node by separation, which allows for expanding the most relevant node. This is said to be a natural way to solve the issue of stopping, as well as addressing the *expand next* issue. [14]

By using a bayesian game tree search, the returned score by the evaluation is a probability distribution. It represents the expected change in score if the corresponding node has been expanded during a search. With this method, there are possibilities for training the evaluation function to return probability functions.

Junghanns describes several methods for addressing the opponent issue, two of which are *opponent model search* and *speculative play*. The idea of the former is that the agent can model the opponent making it able to predict where it might do mistakes. This introduces two values in each node representing what each player believes the node values are. In short, two evaluation functions are used, making it possible to exploit mistakes by the opponent. Ideally, this assumes that the agent knows the opponent's evaluation function and search-depth. Mannen agrees with this statement but also states that the agent needs to have better chess knowledge than the opponent. In short, opponent model search is about finding the best move in respect to the chess knowledge of the opponent. This approach has also been researched in other domains, such as in the game of tic-tac-toe [18]. It is *believed* that agents can benefit from modeling the opponent and then adapt their strategy accordingly, by basing their decisions on the opponent's weaknesses. The idea of the latter approach (speculative play) is to select a move which is not necessarily the best, but one which provides the opponent with *the smallest relative number of optimal replies*. [14, 16, 18]

Junghanns concludes that there is no practical alternative to the alpha-beta search method for computer chess, but also concludes that there were promising ideas with potential in the future. As of 2017, this statement is almost 20 years old. Looking at one of the top open source chess engines, namely Stockfish, alpha-beta seems to continue being the popular approach. This was checked in their source code hosted at github [19] where the *search.cpp* file shows the use of iterative deepening with alpha beta values, although with the complexity of multithreading.

Quiesce search is a method that dates back to 1950 a research conducted by Shannon [7]. This solves the issue of evaluating tense positions, meaning positions where pieces can be captured and exchanged. The problem arises when evaluating a position after, for instance, a queen captures a pawn, making the player believe it is a pawn up. Having searched deeper the agent would have discovered that the queen could be recaptured, having the player exchanged a queen for a pawn. Quiesce search can be executed when the ordinary search reaches the desired depth or makes the depth count increase until the position has no more capturing moves.

This technique ensures more accurate evaluations. [7, 16, 20]

3.4 Belief state

Dealing with problems in environments that are partially observable, a belief state is required. Agent in these environments does not know exactly which state they are in, and thus, needs a form of belief. An agent's belief state is its current belief about what state it might be in, this also means that after performing actions the exact outcome might be unknown. The *belief state space* refers to all possible actual states an agent could be in. After performing actions the environment can generate percepts to the agents, as long it has sensors to perceive them. What these percepts are, depends on the domain and what information that can be useful from the environment. The agent also has to maintain its belief state after performing actions and receiving percepts. This is at the core for any intelligent systems. [21]

Sequential decision problems are those when the utility after performing an action depends on previous actions. A transition model can provide information regarding the outcome for each action of each state. It follows the Markovian property when the next state depends only on the current state. When actions have stochastic outcomes means there are probabilities linked to what exactly might happen. That being said, there is a probability $P(s' \mid s, a)$ for reaching state s' when performing action a in current state s. When the environment is fully observable these problems can be addressed using *Markov Decision Processes* or (MDP). Since the outcome of an action is probabilistic there is a need to specify what the agent should do in any state. This is referred to as the *policy*, denoted $\pi(s)$, which return the recommended action in state s. A policy is generated from the initial state, meaning the history might be different each time it is calculated, which is why a policy is measured by what is expected given all possible states. A policy which yields the highest expected utility is referred to as the *optimal policy*, denoted π^*. In short, and since the agent knows its state, it can execute action $\pi^* s$. However, when the environment is partially observable it becomes more challenging. In addition to the transition model, a sensor model is needed, that being $P(e \mid s)$. It provides with the probability of perceiving evidence e in state s. A key concept with partially observable MDP, more commonly referred to as POMDP, is that the belief state becomes a probability distribution over all possible states the agent can currently be in. [21]

Research conducted by Rodriguez, Parr, and Koller [22] looks at reinforcement learning with approximate belief state related to POMDP problems. They state that finding the optimal policy is PSPACE-hard when it comes to the underlying states, and as of 1999, solving POMDPs are most relevant to relatively small problems, problems which are not trivial when it comes to computing or representing a full

belief state. Having an exact belief state to maintain is impractical for large problems. Instead, they use the concept of *approximate belief state* which is not well described other then it is fairly close to the true belief state. The research compares 3 different approaches for belief state reinforcement learning, 2 with the use of Neural Network (NN) where one uses full belief states, while the other uses an approximate belief state. The third uses a SPOVA method, a method which uses a *function approximator* customized for POMDP problems. According to the results, SPOVA learned faster compared to full belief state NN, which nonetheless, catches up eventually. Another test suggests that the NN with approximate belief state can search much more efficiently since it requires fewer parameters. In general, the paper concludes that, for medium sized problems, approximate belief state reinforcement learning can *outperform* the alternatives when it comes to fewer training iterations and faster training. [16, 22]

Another research by Mannen trained evaluation functions within the domain of chess. He stated that reinforcement learning is a technique suited for solving MDP problems. The research focused on using NN in order to train an evaluation function using multiple games played by experts. The 2 experiments conducted within the chess domain were material balance, and evaluating chess positions. The former tests showed that the networks had a higher expectation of winning when the player have one more bishop rather then a knight compared to the opponent. For the second test focused on material, mobility, central control, and positional pawn structures. 7 different evaluation functions were trained based on 50000 games. All using different networks for the 3 phases of chess, that being the opening, middle game, and end game. The paper concludes that it is useful to divide the training sets into different categories since having different networks for the different phases of the game gained better results compared to only using one network. [16]

3.5 Evaluation function

Material is a big part of the evaluation function, being materially better has been a common indicator of which player is better. This is the sum of each piece from both sides, where each piece type is associated with a base value. The most common values for each piece are pawns being worth 1, knights and bishop 3, rook 5, and queen 9. The king gets usually a big score, such as 200, making the agents discover king captures obvious, discouraging it to be captured by any means.

An evaluation function usually incorporates pawn structure weakness and mobility. Weakness in pawn structure is double, backward, and isolated pawns. Double pawns make it impossible making one protect the other, at the same time one of them are blocked. Backward pawns are not able to advance safely, as no neighboring pawns protect it. Isolated pawns are pawns which have lost their neighbouring

pawns, making them vulnerable. Mobility on the other hand, is basically the number of legal moves a player has in a position, meaning the player with most moves has a slight advantage. [23, 7, 16]

Even though each piece type has a base value it's strength also depends on where it is placed on the board. This usually depends on how many important squares a piece protects. One technique to encourage placing pieces on good squares are with *piece-square tables*. A table here contains information about additional points for each square on a chess board. Since each piece type is unique on how they move on the board, they all get a different table. This means that squares that are good for a rook, not necessarily are good for a bishop and vice versa. These tables also reflect what color is being played. For instance, pawns get a higher value the longer it moves to the other side, getting closer to be promoted. This table would only be relevant when played as white, meaning each table needs to be transformed accordingly to get the same effect when playing as black. [24]

The awarded master thesis conducted by Michniewski, looked at generic algorithms, heuristic searching, and machine learning. Unfortunately, the thesis is written in Polish. This made it necessary for an Associate Professor at NTNU Gjøvik to assure its content is legit. He suggested making contact with the author to get more insight if there has been any further research in the field. After exchanging emails with the author, a more general idea of the work was obtained, also referring to the chess-programming wiki for further information.

The thesis focus on investigating how chess engines perform when mainly relying on piece square tables. An evaluation function was developed with designed piece-square tables, and numerous chess engines were compared, including his own named Tytan. The experiments showed that strong programs became weaker without their original evaluation function. For instance, the engine Crafty, which is originally stronger than Tytan, became weaker in comparison. The conclusion was that the evaluation function is the key to success, and not only relying on piece-square table. Other engines also use the piece square table approach such as Gerbill and Rebel. However, as stated on the wiki "*Material and piece-square tables alone are enough for a program to play a semi-decent chess*". [25, 24]

3.6 Chess Rating Systems

Chess rating systems have been an important factor to promote chess, making tournament chess more popular. The official rating system as of 2017 is a variation of the *ELO system* which was developed in the 1950s. According to a research paper conducted by Glickman and Jones [26], the system has been adopted and modified by numerous national chess federations ever since. It is believed that each chess player possesses an unknown current playing strength, which is an assump-

tion the ELO system is based on. This *strength* is estimated based on the ratings of the competing players. There are numerous practical uses for a chess rating system such as tournament pairing, tournament sectioning, and prize eligibility, as well as in qualifying systems for elite tournaments and, not to forget, the ability to monitor one's own progress as they play chess. [26]

Unfortunately, the ELO system has its flaws. After USCF converted to this system multiple modifications have been tested. One issue is dealing with new players as the ratings are only based on a few games. These are referred to as provisional ratings. Glickman and Jones describes an issue with the provisional ratings were winning a game against a lower rated player can result in losing rating. This is an issue since winning a game is never evidence that a player is overrated. This is addressed by a adding a condition to these scenarios. [26]

How much rating is taken depends on a development coefficient, also referred to as the K-factor which is provided in the Equation 3.1. It represents the maximum change in rating, which depends on the outcome of a game, but also each player's current rating.

$$\mathrm{NewRating} = \mathrm{Rating} + \mathrm{K} \times (\mathrm{Result} - \mathrm{Estimate}) \tag{3.1}$$

The *Result* is 1 if the game ends in a win, 0.5 if drawn, and 0 if lost. The estimate represents what outcome to expect based on the agents current rating, which is displayed in Equation 3.2. With two agents having the same rating, the estimate becomes 0.5, meaning that an agent is expected to win 50% of the time. Rounding the estimate and adding it to the current rating of the player gives the expected rating.

$$\mathrm{Estimate} = \frac{10^{\mathrm{Rating}/400}}{10^{\mathrm{Rating}/400} + 10^{\text{Opponent Rating}/400}} \tag{3.2}$$

Players with a rating of 2400 or higher get a lower K-factor since their skills are not believed to change as much as new players. In other words, their rating is more stable. However, this is purely based on players rating. According to World Chess Federation (FIDE), new players to the rating list gets a K-factor of 40 until 30 games have been completed in events. From that point on the player gets a K-factor of 20 until they reach a rating of 2400 from which it will settle on 10 [27]. One paper acknowledges the issue with determining a proper K-factor and suggests the number of games played could influence this decision. [26]

Another research paper by Bester and Maltitz [28], suggests adding the concept of momentum into the ELO system. They acknowledge its usefulness in two player games, including team-based games and games with score outcomes. Three different systems are tested and experimented, namely the Buffer, Switching Mo-

mentum, and Deficit system. The Buffer system builds up momentum which is later used to offset the rating. The Switching Momentum system adds a momentum value of 1, 0 or -1 to each player depending on whether they are on a winning streak, broken streak, or losing streak. The last system, adds a deficit condition when a streak is broken; the player's rating will not change until the deficit is recovered.

Each system is compared against the original ELO system based on two players with different initial ratings. In the first 2 systems mentioned, the true ratings are obtained faster. The deficit system was almost in line with the original system, suggesting it could be a more effective system as it does not have all the same weaknesses. It is concluded that momentum can be included making the ELO system more effective. [28]

The chess-programming wiki page describes different methods for comparing the strength between chess engines. The suggested tests are t-tests and likelihood of superiority (LOS). The former can tell whether two chess engines are different in strength by comparing their ratings after played games. One can also test whether one is stronger than the other, by using a one-tailed test. The latter is a simple calculation according to the source which refers to work done by Remi Coulm, a French associate professor in computer science, and Kai Laskos, a computer chess and Go theorist. The provided, and simplified LOS equation is:

$$\text{LOS} = \frac{1}{2} \times \left[1 + \text{erf}\left(\frac{\text{wins} - \text{losses}}{\sqrt{2 \times (\text{wins} + \text{losses})}}\right)\right] \tag{3.3}$$

The erf is referred to the standard C/C++ library, namely the error function *std::erf*, which is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \times \int_0^x e^{-t^2} \tag{3.4}$$

Both ELO rating and the LOS calculation seem to be an approach for distinguishing between the strength of chess engines. The second best chess engine (as of May 2017) is also open source [29]. This is Stockfish 8 where the community has provided a testing framework for simulating games between different versions in order to distinguish between their strength. This framework is called *fishtest* where anyone can contribute and view results. The community provides with regression tests displaying the increase in ELO rating points between different tests. The site which provides the top chess engines, also refers to the LOS value, along with rating to indicate which engine is superior.

4 Implementation

In order to investigate the game of darkchess and how the uncertainty affects the domain a working application was needed, able to perform both AI vs AI testing, but also to conduct user testing. The focus of the latter was intended in understanding how people think and act when trying to solve the problems that darkchess present. There exist no database of darkchess games, or any open source web projects at the time that could help and guide the starting point of this research. This led to the decision of developing a working web application mostly from scratch.

4.1 Development setup & tools

With a web application, one can potentially reach out and collect more games from people enjoying chess. However, this approach also implies concerns related to exploits, leading to the decision of having the AI logic on a server, assuring more trustworthy results. The reason why the AI and the web application was not on the same server was mainly that of already owning a web hotel at one.com. By using their tools would save time compared to setting up a web server, and at the time of development one.com only offered PHP server-side development. Since javascript were more a familiar programming language than PHP, learning Node.js appealed more, and thus became the technology of the application. With that in mind, the working application was developed in two parts, a web application and a Node.js server for the AI implementation. The web application was only the front end, using *Bootstrap 4* for graphical components, *Font Awesome* for icons, and *chessboard.js* for displaying and interacting with a chess board. The latter had to be modified in order to support darkchess FEN notation. The AI was developed using Node.js which is hosted on an Ubuntu server. For communicating between the web application and the AI, web socket was used, where the npm library *ws* helped in setting up a messaging system. The entire project is under one repository and open source, hosted on GitLab following this link *https://gitlab.com/tmellemseter/darkchess*.

The server-side consists of mainly the 4 files: server.js, darkchess.js, agent2.js, and alphabeta2.js. The server file contains setting up a web socket server and handling different messages being passed from clients, such as *START_GAME* and *ON_DROP*. The former is passed when a player has selected a color to play and pressed start. When the message is received, a game of darkchess is initialized along with the AI opponent. In other words, the server defines a darkchess object

from darkchess.js and the agent from agent2.js. The latter is passed each time the player drops a piece from one square to another. If an illegal move is attempted, the server returns the correct board position along with a feedback message. Thus the server also acts as the referee with the game state being the darkchess object. The darkchess library provides an API to verify, make and undo moves, along with evaluating material on the board and retrieving information about the current state of the game. The agent also has a darkchess object which represents its belief state. In addition, the agent also contain an evaluation function, a random generator, an alpha-beta search function, and information regarding opponent's pieces, and boolean values for enabling and disabling certain features. The agent instantiates an alpha-beta object (from alphabeta2.js) which can define how to search. For instance, the agent's evaluation function is passed to the alpha-beta object. When the agent searches, it makes and undo moves on its belief state. However, this is not an accurate representation, as all opponent moves are registered in this state, even those it should not know about. Before searching, the agent's current information about which squares are visible and hidden is provided which does not change during a search. This way, the agent only searches within its visible space, making sure it does not observe hidden pieces. That being said, it is the *opponent piece information* which is more accurately the agent's belief state. This is the number of visible and hidden pieces for every piece type which is needed to be distinguished when evaluating the material on the board.

4.2 Evaluation function

In traditional chess, the evaluation function looks at the current board position and compare different aspects on the board between the white and black player. It returns a value representing which player is better where a positive value means white while a negative number means black. Some aspects are material, positional weakness, and mobility. Material is the value of all pieces where each piece type is linked to a base value which represents its strength. There are no standard values, but the chess community seems to agree on pawns being worth 100, knights and bishop being worth 300, rooks 500, and queen 900 [23, 7]. In this application, the values selected follows Tomasz Michniewski's idea which is to prevent exchanging minor pieces (bishop and knight) for 3 pawns, encourage the agent to keep the bishop pair and avoid exchanging two minor pieces for a rook and a pawn [30]. Following this guideline the pawns are still worth 100, knights changed to 320 and bishop to 330, while rook and queen keep their values of 500 and 900. There is also another reason for choosing these values as they relate to the use of *piece square tables*, a technique that rewards pieces being positioned on good squares, and punishing them for being positioned on bad squares. For instance, a knight is

better at the center of the board as it will control more squares, and according to the same source, a knight at the center is worth $320+20 = 350$ while one in a corner is worth $320+-50 = 270$. In other words, this technique encourages moving pieces to good squares. Positional weakness relates to the pawn structure, specifically the number of double, backward or isolated pawns, and mobility is simply the number of legal moves.

4.2.1 Material

In darkchess, the opponent's material is known, although not necessarily their positional strength, while pawn structure and mobility becomes challenging to calculate. Not knowing the exact position of the opponent's pieces, the average of their possible square values was used. Unfortunately, pawn structure weakness, and mobility were omitted, meaning the evaluation function only considers material. The evaluation function is defined as:

$$Score = \alpha \times \left(\sum_{i=0}^{n=6} (TotalValue(W, i) - TotalValue(B, i)) \right) \times whoToMove \quad (4.1)$$

Where $\alpha = 0.01$, the centipawn constant. The value is scaled down such that a pawn with the base value of 100 becomes 1, which translate to white being up *one* pawn. *whoToMove* is 1 if the player to move is white, and -1 if black. This is important when using the alpha-beta negamax framework, returning the score relative to the player in turn. The $TotalValue(p, t)$ returns the total value of all pieces of piece type t for player p where t is either pawn, knight, bishop, rook, queen, or king. This is also the list that is summed in Equation 4.1, hence n=6. $TotalValue$ is the sum of the visible pieces and the average of the hidden pieces, getting $TotalValue(p, t) = VisibleValue(p, t) + HiddenValue(p, t)$. Both of these functions are defined as follows:

$$VisibleValue(p, t) = N \times BaseValue(t) + \sum_{i=1}^{N} SquareValue(p, t, i) \quad (4.2)$$

$$HiddenValue(p, t) = N \times \left(BaseValue(t) + \frac{\sum_{i=1}^{N} SquareValue(p, t, i)}{N} \right) \quad (4.3)$$

Note that, $N = |t|$, the number of the respective piece type, either hidden or visible, making the summation over N the average positional score per hidden piece. Bishop is however treated differently as they can only move on light or dark squares. In short, the summation is relative to the game state and piece type.

4.2.2 Board Representation & Piece-Square Tables

The $SquareValue(p, t, i)$ represent the piece square table of piece type t from player p's perspective. The i is used to get the table index of the ith piece p on the board. All 6 piece square tables are stored as static 2-dimensional arrays in the agent2.js file from white's perspective. For black, the tables has to be transformed by flipping it horizontally and vertically as not all tables are symmetrical. Also note that, if the agent play as white, and is calculating its own material, $HiddenValue$ should return 0 as the agent will always have full knowledge about its pieces.

The board is represented as an array of size 128, twice as long as the number of squares on an 8x8 chessboard. Only half of the array are valid squares, starting at index 0 which represent square a8 and h8 would be at index 7. The next square in the array, however, being a7, is not at index 8, but 16. In other words, every second row (8 indices) are empty. This square-centric board representation is called *0x88* [31]. The reason behind this is that detecting *off the board* squares when generating legal moves become cheaper and faster, due to bitwise operations. This method is used in the chess.js library which darkchess.js is heavily based on. That being said, the summations illustrated in the equations are in code iterating over the entire board and then filters what information that is important. For instance, for only considering visible squares, the *in view information* is used. It has the same data type as the board, but with a true or false value for whether the corresponding square is visible, which is updated every time the agent does a move. When iterating, the current square is checked against the in-view information, if it is not in view, a *continue* is triggered, meaning it immediately jumps to the beginning of the loop and starts checking the next square. For the piece square tables, it means they need to be converted to the *0x88* data structure.

4.2.3 Searching

When considering what move to do the agent searches in a MiniMax fashion. The algorithm used is a modification to the alpha-beta pruning, which works such that the agent considers all of its own moves to begin with, before considering all of the opponent's responses. This is a search with 2 in depth. After picking two moves, the board is evaluated and given a score indicating whether the position favors one of the players. After analyzing every response by the opponent the agent can conclude, based on the score, that the best the opponent can achieve, is for instance, an equal position. The next move the agent considers, one of the opponent's move is winning a rook, meaning no further opponent moves need to be considered, as the first move resulted in an equal position. Analyzing the first move creates a *lower bound*, meaning anything worse can be rejected. When going deeper then 2, both players can affect the outcome resulting in an *upper bound*. In other words,

a player's lower bound is the opponents upper bound. It is these bounds that are referred to as *Alpha* and *Beta*.

There are different approaches in implementing alpha-beta pruning, where it seems that a recursive solution is popular. One can implement it having one recursive function, but having two distinct indirect recursive functions for the max and min player is also possible. With the negamax framework, one simplifies this by merging those two into a single function. This is possible due to the mathematical relation $max(a, b) = -min(-a, -b)$. The main reason for the negamax framework was used is mainly because the chess programming community provides with pseudo code on how it is done in addition to the *Quiesce search*. [15, 32]

The modification is to continue searching when the list of possible opponent moves are empty. This is shown in Algorithm 1, when the move list is empty, one simply swap the player to move and go directly to the next depth. Another change is that a depth of 0 is passed when starting the search, instead of the desired depth. The max depth value can be set from the agent's object, which is then passed down to the alphabeta object. A search is started by calling

$$alphaBeta(-Infinity, Infinity, 0, principalLine)$$

The last parameter is an empty array which will contain the principal line when the search has finished. The array contains the list of moves by both players the agent believe are the best, where the best move to do is at index 0. This array is updated each time a better alpha is detected. However, when different agents play against each other, another array stores multiple move variations to consider. This is shown at the bottom of the algorithm when the recursion has return to its first instance, meaning back to when depth is equal to zero. Remember that, when using the negamax framework, the evaluation function needs to returned the score from the players perspective, which is shown in the Equation 4.1. The agent object provides a *move* function where no parameters are needed, it gets a list of moves with their corresponding score from the alpha-beta search and selecting a move is based on a distribution which is calculated from their score. This distribution is the

score normalized ensuring that good moves are more likely to be selected.

Algorithm 1: AlphaBeta

```
Input: Alpha, Beta, Depth, PLine
Output: Score

Line = EmptyArray

if Depth reached || Game finished then
    Score = Quiece(Alpha, Beta, PLine)
    Score = (Score − GetRisk())
    return Score
end

Moves = GetMoves()

if Movesareempty then
    Swapplayer
    Score = −AlphaBeta(−Beta, −Alpha, depth + 1, PLine)
    Swapplayer
end

for Move in Moves do

    DoMove(Move)
    Score = −AlphaBeta(−Beta, −Alpha, Depth + 1, Line)
    UndoMove(Move)

    if Score >= Beta then
        returnAlpha
    end

    if Score > Alpha then
        Alpha = Score

        PLine[0] = Move
        for i = 1 to Line.length do
            PLine[i] = Line[i − 1]
        end

        if Depth == 0 then
            PrincipalVariations.push(move, score)
        end
    end
end

return Alpha
```

Quiescence search is applied to make sure the position to be evaluated does not include any tactical moves. That is moves which immediately change the material balance, such as captures and promotions. The reason is to avoid evaluating at the desired depth since the last considered move can, for instance, be a queen capturing a pawn, which will grant a +1 to the score (if white). However, has the search gone one step deeper, it could see that the queen gets recaptured, making the score -8, a benefit for black. Thus, quiescence search is applied when the ordinary search reaches its desired depth. In a case of no captures, the position is evaluated and the score returned. the implemented function is very similar to the alpha-beta method, using the same pruning technique. After evaluating the position, the method checks whether there is a beta cutoff where the beta is returned, or possible updating alpha. Note that there is no checking against any depth limit, meaning it could potentially run undesirably long. However, the number of capturing moves is usually a fraction of all possible moves, especially in the opening and

endgame.

Algorithm 2: Quiesce

```
Input: Alpha, Beta, PLine
Output: Score

Line = EmptyArray
StandPat = Evaluate()

if StandPat >= Beta then
    return Beta
end

if Alpha < StandPat then
    Alpha = StandPat
end

CapturingMoves = GetCapturingMoves()

for Move in CapturingMoves do

    DoMove(Move)
    Score = −Quiesce(−Beta, −Alpha, Line)
    UndoMove(Move)

    if Score >= Beta then
        return Beta
    end

    if Score > Alpha then
        Alpha = Score

        PLine.push(Move)

        PLine[0] = Move
        for i = 1 to Line.length do
            PLine[i] = Line[i − 1]
        end

    end
end

return Alpha
```

4.3 Belief state

Making the agent able to evaluate its position it needs to know which state it is in. This is not trivial in partially observable environments, as it cannot know for sure what the reality is within the unobservable space. This means the agent needs to consider the state it *believes* it is in. Before any move has happened, the agent's belief state is the actual state, a copy of the initial board position. The information in the state is a board representation, which squares are visible and not, king placement, the player to move, castling rights for both sides, number of half-moves, the move number, and a list of all moves that has happened so far in a game. This is referred to the *game state*, and the referee has the exact same game state structure. In addition, specifically for the agent, the agent needs to keep track of the opponent's pieces, meaning how many of each piece type are visible and hidden.

When a game has started and as moves happen over the board, the agent register all move types as specified in Table 1, even the unknown ones. This might sound like cheating, but this information is never used when evaluating the position or is revealed during a search. This method makes updating the piece type information trivial as it boils down to increasing and decreasing how many are visible and hidden. For example, imagine the opponent has done an unknown knight move which becomes registered in the agents game state. After the agent responds, having chosen a move which reveals the unknown piece, it simply decreases the number of hidden knights by 1 and increases the number of visible knights by the same amount. This happens for all pieces that get revealed, which can maximum be 7 pieces.

4.4 Assessing Risk

In traditional chess, one can never be 100% sure of what move the opponent might do. Even when a player allows the opponent to win a pawn. This is known as a gambit in chess. Take for instance the opening *1. e4 e5, 2. d4 exd4. 3. c3* where white allows black to capture 2 pawns. In other terms, White sacrifices 2 pawns in order to achieve rapid development with attacking possibilities. However, one cannot know for certain whether black will accept this line known as the Danish Gambit or decline it. That being said, there exists some probability whether the opponent attacks a piece given a position where it is possible to attack. This is defined as:

$$\mathrm{P}(\text{Attacks} \mid \text{Can attack}) = \mathrm{P}(A \mid C)$$

Unfortunately, this is challenging to calculate as it depends on who is playing. The probability $\mathrm{P}(C)$ in standard chess is always known and is therefore either $\mathtt{true}$ or

$\mathtt{false}$. This is because one has complete knowledge over the board, but in darkchess this is not the case. However, knowing the probability of the opponent *can* attack a certain square on the board is not that helpful, since attacks can be both good and bad. To determine the difference between good and bad attacks depends on both the attacking and captured pieces. Exchanging a piece by a less valued piece will lead the attacker being up in material. This is preferred, although there exist exceptions where a piece can be positionally stronger than a higher valued piece. There are also discovered attacks. Those occur when a piece moves out of the way for another piece which then attacks. This means that the discovered move can almost allow any capture, being it good or bad materialistically, as long as the discovered attack threatens a more valuable piece. Regardless, a more useful probability is $P(\text{Can attack piece x with a less valued piece y}) = P(C_{x>y})$. In traditional chess, losing a quality can be considered a blunder, meaning a huge mistake that can end in a loss. Here, *losing a quality* is defined by exchanging a high valued piece against a lower valued piece. This should not be confused with *losing material* as that include sacrificing pawns which might have its advantages as described above. In short, since the material is such an important factor when evaluating which player is stronger, one should always exchange any low valued pieces against any higher valued pieces. This defines the assumption:

Assumption. *In a position where the opponent can capture a piece with a lower valued piece, the attack will happen.*

$$P(A_{x>y} \mid C_{x>y}) = 1 \quad \textit{and} \quad P(A_{x<y}) = P(C_{x<y})$$

That being said, the point of assessing the risk is to manipulate the evaluation score based on how much risk of losing quality is present in the position. This risk is calculated when the ordinary search algorithm has reached the desired depth. This means that risk assessment is done after the agent has made $(\mathtt{depth}/2)$ number of moves where $\mathtt{depth}$ is an even number. This becomes inaccurate for depth greater than 2 as the risk will be evaluated after multiple moves have been made, disregarding the risk of the position after the first move. However, since the priority has been looking with a search depth of 2, this implementation did not become an issue. That being said, this is also the reason risk assessment is not considered when doing quiesce search. The other reason for not going beyond a depth of 2 is due to the lack of optimization, meaning at greater depth the search time increase exponentially.

When assessing the risk, the agent looks at its own pieces and calculate the probability of being lost by a less valued piece. Some pieces have multiple less valued pieces to consider such as the bishop having both pawns and knights being

less valued. Table 2 shows this relation for all piece types.

Piece type	Less valued pieces				
Pawn					
Knight	Pawn				
Bishop	Pawn	Knight			
Rook	Pawn	Knight	Bishop		
Queen	Pawn	Knight	Bishop	Rook	
King	Pawn	Knight	Bishop	Rook	Queen

Table 2: Less valued pieces

The total risk in a given position is calculated as follows:

$$\mathtt{TotalRisk} = \sum_{i=m}^{n} (\alpha \times \mathtt{BaseValue}(m) \times \mathtt{PieceRisk}(m)) \tag{4.4}$$

Where n is the number of own pieces on the board, where m is the first piece. As the equation shows, the risk is weighted against the base value of the given piece and scaled by the centipawn value α. The evaluation score is decreased by the this $\mathtt{TotalRisk}$. Since, the king has a base value of 20000, even a small percentage grants a big decrease in the score, making the agent scared by exposing the king. The risk assessment for any piece is defined by the probability of *at least one less valued piece* captures said piece. The equation is as follows:

$$\mathtt{PieceRisk}(m) = 1 - \prod_{i=t}^{|l|} \prod_{j=0}^{|t|} \left(\frac{total - possible - j}{total - j} \right) \quad \text{where total is} > 0 \tag{4.5}$$

Note that, l is the list of *less valued piece types* in relation to m, where t is the first piece type in the list. The inner product notation represent the probability of no pieces of piece type t that can attack m. *Total* and *possible* represent the number of dark squares where *possible* depend on piece type t and what square piece m is placed on. This is based on the Assumption 4.4, which makes it possible to calculate the probability of losing quality, specifically meaning $P(C_{x>y}) = \text{possible/total}$. Note that, for bishops, the denominator is not the total number of dark squares, since a bishop can only move on either light or dark squares. In other words, $total$ depends on piece type y.

5 Method

5.1 Comparing different AI implementations

A node script was developed in order to make 2 agents play darkchess against each other. The script can be found under the generate folder in the repository, with the file name generate2.js. It takes two arguments, a number for how many games the agents will play, and a file name to save the test. After executing the script and the test has finished, the result is stored in a CSV formatted file, able to be opened by other programs such as R. The first line in the file, however, is a random seed used in the random number generator for the agents, where the second is the header information, namely the column names. There are a total of 9 columns: an *id* (which specifies the game number), a *score* (given by the referee), the agent's name (labeled *player*), which *color* it played as, the *result* of the game, the ELO *estimate*, and current ELO *rating*, and at the end, a boolean whether the agent started to miscalculate the position (labeled *error*). In a case of a draw, information is also provided for what triggered it (labeled *comment*). A draw can be reached in two ways either by the *50 move rule* or *insufficient material* which occur when only the kings are left on the board. A row represents one agent's data, meaning one game adds two rows. Table 3 show an example of this structure.

By passing a filename as the first argument instead of a number will repeat the test of the passed file, making each test repeatable. Unfortunately, the settings for each agent is not stored in each file and thus has to be manually changed before repeating a test.

To begin with, each agent is assigned an ELO rating, which indicates their strength, and for all tests, these are set to 500 for both agents. This is to observe whether one of the agents will increase in rating, indicating a better agent. Since

id	score	player	color	result	estimate	rating	error	comment
187	-218	agent1	w	0	565	537	false	
187	-218	agent2	b	1	436	463	false	
188	9	agent1	b	0.5	538	533	false	50 move rule
188	9	agent2	w	0.5	463	467	false	50 move rule

Table 3: An excerpt of a .csv file generated by the generate2.js script

chess is a zero-sum game, when one agent wins a game, it will take a few rating points from the losing agent, meaning the total rating will always be 1000.

The LOS is also calculated between each test, getting a sense if any agent is superior. This is calculated using the Equation 3.3. However, Equation 3.4 is not supported directly in R, instead an equivalent error function is used, being Equation 5.1, where pnorm is a distribution function already available in R:

$$erf(x) = 2 \times \text{pnorm}(x \times \text{sqrt}(2)) - 1 \tag{5.1}$$

A total of 12 different tests have been conducted, each with different settings between the agents. For each test, a 1000 independent games have been played between the agents, in order to observe trends and analyze their difference. All tests look at the same data, which is, any advantage of playing a specific color and looking at rating trends over the duration of games.

The different main features an agent can enable are risk assessment and move ordering. There have also been experimented with different search depth and reducing the severity of risk. This leaves a total of 10 main tests, where each agent's setting is specified in Table 4.

Filename	Agent 1		Agent 2	
	Risk	Move Ordering	Risk	Move Ordering
nvsn.csv	FALSE	FALSE	FALSE	FALSE
rvsr.csv	TRUE	FALSE	TRUE	FALSE
mvsm.csv	FALSE	TRUE	FALSE	TRUE
mrvsmr.csv	TRUE	TRUE	TRUE	TRUE
rvsn.csv	TRUE	FALSE	FALSE	FALSE
mvsn.csv	FALSE	TRUE	FALSE	FALSE
mvsr.csv	FALSE	TRUE	TRUE	FALSE
mrvsn.csv	TRUE	TRUE	FALSE	FALSE
mrvsr.csv	TRUE	TRUE	TRUE	FALSE
mrvsm.csv	TRUE	TRUE	FALSE	TRUE

Table 4: Agent settings for each main test, all searching at a depth of 2

An R script has been created in order to plot the data of the .csv files to present the results graphically. This makes it fairly easy to reproduce the graphs for both old and new tests. After loading the *tests.R* script into an R console, all plot functions becomes available, including a *loadFile* function for loading one csv file by passing its name. It will return a data frame which can be given as input for most plot functions.

First of all, the data is checked if they follow a normal distribution. This was done using the Shapiro-Wilk normality test in R against an agent's change in rating.

File Name	Shapiro-Wilk	Anderson Darling	H0
nvsn.csv	0.02715	0.1355	Contradiction
rvsr.csv	0.0006014	0.001721	Reject
mvsm.csv	4.8×10^{-5}	0.0003627	Reject
mrvsmr.csv	0.08866	0.3862	Fail to reject
rvsn.csv	0.06964	0.2416	Fail to reject
mvsn.csv	0.001735	0.04311	Reject
mvsr.csv	2.2×10^{-16}	4.7×10^{-12}	Reject
mrvsn.csv	0.002038	0.002055	Reject
mrvsr.csv	1.4×10^{-12}	7.4×10^{-11}	Reject
mrvsm.csv	0.01021	0.11714	Contradiction

Table 5: Normality test based on rating change over games for Agent1

The other agent's result would be symmetrical because of the zero-sum property of the game. Another method is also provided for comparisons, namely the Anderson-Darling test which is provided with the *nortest* package in R. This introduces the following hypothesis, both reject H0 when the p-value is less than 0.05.

$\mathbf{H_0}$: The data is normally distributed

$\mathbf{H_1}$: The data is not normally distributed

The results for each test are shown in Table 5, including both normality methods with their corresponding p-value. The last column provides with the result, rejecting or fail to reject H0. A third alternative is also possible when the two methods disagree. These are labeled *contradiction*.

The result shows different assumptions about the distribution in rating for Agent-1. Most tests reject H0, suggesting the distribution is not normal. Two tests fail to reject H0, in which a normal distribution can be assumed, while in two other tests the p-values contradict each other. To further analyze the distribution qq-plots are provided in order to graphically interpret the data. This has been provided by the *qqnorm* function in R. The Figure 5 shows the first 4 tests in Table 5 where all agents are similar when it comes to enabled features. The remaining tests are shown in Figure 6.

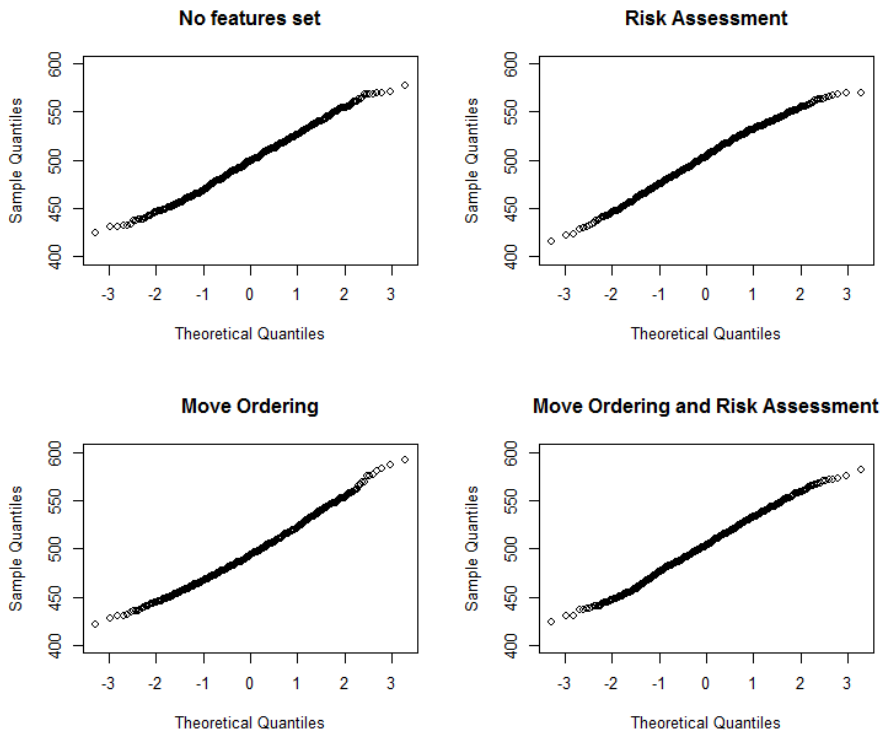


Figure 5: QQ plots for all tests where the agents share the same features

Looking at the QQ plots in Figure 5, the lines all leans towards a *light tailed* distribution. However, they look close to a normal distribution. The *Move Ordering and Risk Assessment* suggested a normal distribution since both normality tests failed to reject H0. Considering the p-value from Shapirov-Wilk, the p-value were only slightly above 0.05, which might suggest a type II error; wrongly retain the possible false null hypothesis. Based on the contradiction for the *No features set* test and the QQ plot, it looks like the Anderson Darling is also a type II error.

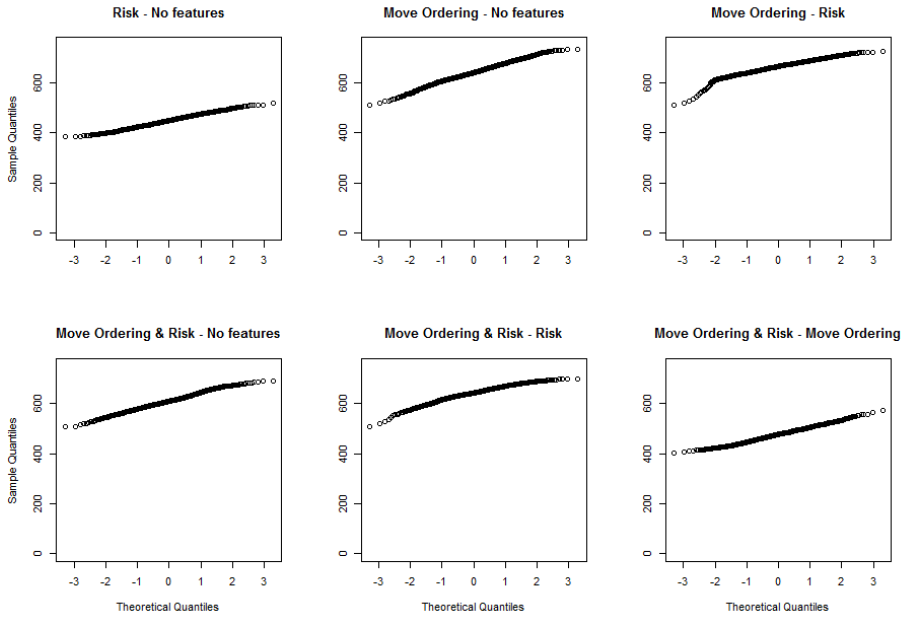


Figure 6: QQ plots for all tests where each agent are different

The first QQ plot in Figure 6 has a slight *light tailed* distribution characteristic based on the endpoints. The p-value provided by the Shapirov-Wilk test is slightly above the threshold 0.05, suggesting another type II error. Both *Move Ordering - No features* and *Move Ordering & Risk - Risk* seem to have a slight *left skew* distribution. Based on the graph of the *Move Ordering & Risk - Move Ordering* test and the contradiction between the normality tests, made it difficult to assess any characteristic of its distribution.

What can be concluded of these tests are that most seem to follow a non-normal distribution. Because of this claim, future tests should not assume a normal distribution. Testing for independence the chi-square test is appropriate, first of all since the data set is large, but also because the data set is synchronous, meaning if any dependent variables exist would also indicate an advantage. For regression analysis, it means a nonparametric regression test is needed.

The Pearson's chi square test of independence is used to observe if two variables are statistically independent. This has been checked between *game results* and *player color*, observing if there is any advantage playing as either white or black. The same method is used between *game results* and *agent*, observing if there is any difference between the agent's strength. This is possible to infer as the re-

sults are symmetrical. Note that, there are 4 different tests which compare agents with the same features enabled, these are: no features, only assess risk, only enable move ordering, and both enabled. In general, the null and alternative hypotheses for all tests with variable X and Y become:

$\mathbf{H_0}$*:* X and Y are statistically independent

$\mathbf{H_1}$*:* X and Y are statistically dependent

The remaining 2 experiments, both increasing depth and adjusting the severity of the risk assessment has been investigated. The risk can be scaled with a factor, meaning a 1.0 keeps the calculated risk while 0.0 is equivalent of not enabling risk assessment, apart from doing unnecessary calculations. Two experiments were conducted on risk sensitivity, where one agent only enables risk assessment with the default 1.0 in severity. The other agent had 0.5 and 0.1 in the respective tests. Only one test was conducted for checking whether increasing the search depth for one agent would improve the results. Here, both agents have only enabled move ordering where one agent searches with a depth of 4, while the other with 2. The reason both has move ordering enabled is to increase the number of cutoffs, making each agent spend less time searching since with an increase in depth the amount of time searching would increase exponentially nonetheless. The limit for a draw according to the 50 move rule was also decreased to 8 to speed up the process, and made it possible to generate 1000 games within a reasonable timeframe. This does not have any impact on the result since whenever both agents start repeating moves, it does not matter how many times they repeat the position before it becomes a draw.

When testing for normality, all three data sets rejects H0. The one which shows a *contradiction* when comparing the p-value between Shapiro-Wilk and Anderson-Darling For this, a QQ-plot is provided, suggesting a slight *light tailed* distribution as displayed in Figure 7.

Regression tests were conducted in order to find trends in the change in ELO rating of the different agents. Only the *no feature* test and *move ordering vs risk* has been analyzed. Since the data related to rating change and played games suggest a non-normal distribution, nonparametric regression analyses have been chosen, specifically the Nadaraya–Watson kernel regression. Different bandwidth has been tested in order to avoid overfitting. In addition, the R's *stat_smooth* function provided in the ggplot2 package uses a Generalized Additive Model (GAM) as default when the dataset is equal or larger than 1000.

In the end, 12 games were more closely examined in order to better interpret the results. This regarded specifically color win-ratio for no features enabled, only risk enabled, and one with move ordering against no features enabled. The games can be found in Figurine algebraic notation (FAN) in Appendix A.

5.2 User Testing

One of the main reasons for conducting the user tests were to get a better understanding on how the agent played against people. Another reason was to cover how people perceive the game of Darkchess, answering questions related to their thought processes, and how they played the game with a focus on risk assessment.

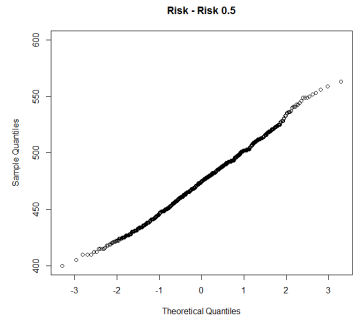


Figure 7: QQ plot for risk vs risk with 0.5 in severity

One requirement for the participants was that they already had knowledge about chess. This was met with all 6 people that participated, which were mainly from school who showed an interest in chess. The agent's settings had enabled risk assessment and move ordering, as well as, only selecting the best move. Each participant played two games, one as white and one as black. Move ordering has been proved important when testing different agents against each other. Enabling risk, on the other hand, suggested the opposite. However, the idea was to analyze if any dark pins or possible double attacks were made, which makes the risk assessment important. Since each participant only played two games the agent could just as well perform at its best knowledge, since the participant would not get familiar with its play style. This concerns especially in the opening, as the agent is not influenced by any opponent moves.

Each test was conducted locally and thus separately. The effort to upload it online was not deemed important as the goal was to perform a qualitative study, analyzing each game more closely, than collecting more data and analyze the sample in relation to draw conclusions about the population. Having it locally also means the ability to observe the participants on how they perform, and can more easily answer questions they might have.

After having gone through the formalities and explaining the rules of Darkchess, the participant answered a pre-questionnaire about their traditional chess knowledge. This covered how much chess they have played during the last month, how much knowledge they had on chess openings, chess tactics, and playing positional chess. A 7 point scale were used to the last three questions indicating their knowledge from *none* to *expert*.

Each participant got a unique ID to link their answers and games, but also to identify their results anonymously. Every participant with an odd numbered ID

played white in their first game and black in their second, while the other played black first and then as white. This was done to prevent biased results in case any comparison between games and color were to be conducted.

Starting a game for user testing, the node.js server script was used. However, in order to save games, it was extended with the ability to access the file system using the npm package *fs*. This was achieved by passing two arguments before starting the script. The first argument is the folder name for which all games by the participant will be saved, basically naming it using the *participant ID*. The second argument is the file name of the game to be played. The convention that was used was specifying the game number followed by what color the participant played as. For instance, the file name *game02white* indicate that the participant played as white in the second game. When a game is finished a .pgn file gets created in the specified test folder. A backup file is also created in an already created backup folder in case any error should occur. The backup files require a unique file name which ended in containing the participant id, game number, and color, followed by a unique string created using the npm package *uuid*. The needed information is also logged in the CLI should there be any errors in creating either of the files.

The PGN format is a standard format for storing all moves from both sides, making it possible to be read from other programs such as Tarrasch, a chess program for windows. Unfortunately, that means no information about what squares are visible and not from both sides and on every move, are stored. Developing the application to load these games became more work than anticipated, and thus down prioritized. First of all, the client side chessboard.js library did not support loading PGN content directly, only loading the board by passing a FEN string. This meant that the correct FEN was needed to be passed from the server each time the board should be updated by either moving back or forward in the game history. Secondly, the algebraic move notation that is used in the PGN needs to be disambiguated. For instance, the move *Rf1* or *Ne4* does not say which rook or knight it is, which has to be specified when both rooks or knights can move to f1 or e4 respectfully. The disambiguation was omitted to prevent revealing too much information when passing the observed moves in the application when playing. In other words, a form of conversion needed to be done.

Before going through each game, the summary of both the pre and post questionnaires were analyzed. Getting a notion of the participants believed chess knowledge regarding chess openings, tactics, and on playing positional chess. The summary of the post questionnaire would cover how the participants played, and which phase of the game they found most challenging and easy, and why.

To analyze the data, Tarrasch was used going through each game, move by move. Unfortunately, Tarrasch does not allow stepping through illegal moves ac-

cording to traditional chess rules. This was, however, only the last couple of moves, except from one game. This was analyzed using the tool available at nextchessmove.com, which also supports loading PGN files. Any interesting positions were noted, like positions where the agent blunders, but also where it acts well.

6 Results

6.1 Agent vs Agent

6.1.1 Test for independence

In the first test, both agents do not have any features enabled. The win ratio conditioned on *player color* and agent including the *error in evaluation* is displayed in all tests starting with Figure 8. This also applies to the error bars, showing a confidence interval of 0.999. Similar results are shown when both agents have only risk assessment enabled, which is displayed in Figure 9. The corresponding LOS calculations for Agent1 for these two tests show 0.39 and 0.73 respectfully, using the Equation 3.3. This suggests that Agent2 is more likely superior in the first test, while the opposite can be said for the second test.

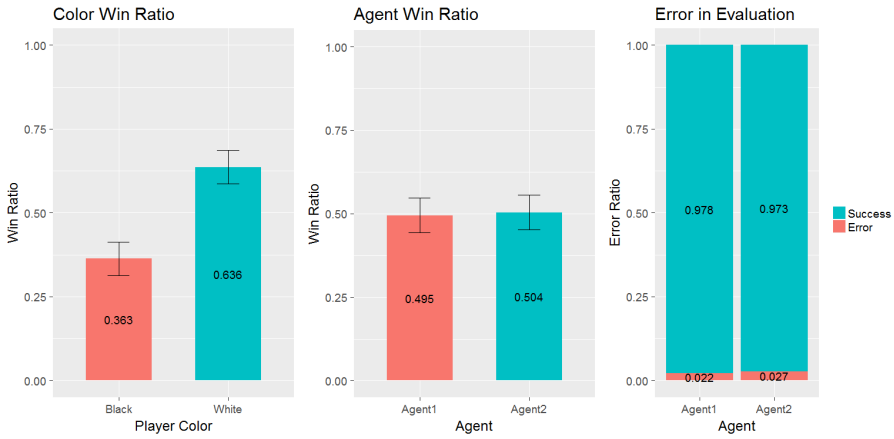


Figure 8: Both agents with no feature enabled, showing error bars with a 0.999 confidence interval

Based on the Pearson's chi-square test, the *p-value* for color win-ratio is 2.2×10^{-16}. It strongly suggests that game results and player color are dependent, meaning the H0 is rejected. In other words, an agent playing as white is more likely to win, given both agents have no features enabled. However, for the next graph in the figure (agent win ratio), the test returns a p-value of 0.92, and H0 fails to be rejected as it is not smaller than the confidence interval. This suggests that there is

no difference between these two agents. One can also observe that the number of games in which an agent starts to miss evaluate a position is small, this is shown for all tests.

Figure 9: Both agents enabled risk assessment only, showing error bars with a 0.999 confidence interval

In the next test, both agents have enabled move ordering only. This test is displayed in Figure 10 with the same graphs. The result, however, shows a change in the color win ratio. The p-value shows the same 2.2×10^{-16} with a confidence interval of 0.999, which also suggests a strong relationship between player color and game results, thus H0 is rejected. Although, with only enabled move ordering, black is more likely to win. For *agent win ratio* the test returns a p-value of 0.33, which fails to reject H0. This has a probability of 0.001 for being wrong. On another note, the LOS calculation shows 0.15 for Agent1, suggesting that Agent2 is almost superior. Similar results can also be observed when both risk and move ordering are enabled for both agents. This is displayed in Figure 11. A difference to notice is the p-value for color win ratio is increased to 6.4×10^{-6}. However, the H0 continues to be rejected. The same conclusion is made for *agent win ratio* with a p-value of 0.48, continuing in failing to reject H0. The LOS, in this case, show 0.80, similar to the previous test, although, here it is Agent1 which is almost superior.

Figure 10: Both agents with only move order enabled, showing error bars with a 0.999 confidence interval

Figure 11: Both agents with risk assessment and move ordering enabled, showing error bars with a 0.999 confidence interval

Testing different settings for each agent's, it is expected to see more difference in agent win ratio. Risk only (Agent1) versus no featured enabled (Agent2) is displayed in Figure 12. Here, both color win-ratio and agent win-ratio return the same p-value of 2.2×10^{-16}, rejecting H0 in both cases. This highly suggests that risk assessment makes the agent perform more poorly. According to LOS, Agent2 is superior over Agent1 as it returns 1.0.

Only enabled move ordering (Agent1) against no feature enabled (Agent2) is displayed in Figure 13. The chi-square test returns a p-value of 0.43 for color win-ratio, which fails to reject H0. For agent win-ratio on the other hand, gives a p-value of 2.2×10^{-16} with a confidence interval of 0.999. Thus, H0 gets rejected, suggesting a strong relationship between move ordering and game results. Having move ordering enabled gives higher winning chances when the opposing agent has no features enabled. Agent1 is 100% superior over Agent2, the same can be said for the next test as well.

The test for only move ordering enabled (Agent1) up against only risk enabled (Agent2) is displayed in Figure 14. This shows very similar results compared to the previous test, thus drawing the same conclusions regarding rejecting or fail to reject H0.

Figure 12: Agent1 with only risk assessment enabled and Agent2 with no features enabled, showing error bars with a 0.999 confidence interval

Figure 13: Agent1 with only move ordering enabled and Agent2 with no features enabled, showing error bars with a 0.999 confidence interval

Figure 14: Agent1 with only move ordering enabled and Agent2 with only risk assessment enabled, showing error bars with a 0.999 confidence interval

The last 3 tests, Agent1 has the same features enabled. That is both risk assessment and move ordering. Agent2 has no features enabled, only risk enabled, and only move ordering enabled . These are displayed in Figures 15 16 17 respectfully. The first two tests show a similarity in results as color win-ratio fails H0 to be rejected due to a p-value of 0.13 and 0.77. The same goes for agent win-ratio, having the same p-values 2.2×10^{-16}, rejecting H0 and suggesting a strong relation between game results and move ordering, even with or without the risk assessment. These also share the same LOS outcome, Agent1 being 100% more likely superior over Agent2. The remaining test shows different results. For color win-ratio the Chi-square p-value is 2.5×10^{-6} which strongly suggests that game results and player color are dependent variables. The same conclusion is drawn between game results and agent, returning a p-value of 3.0×10^{-8}. The LOS for this last test returns the value 1.5×10^{-5} strongly suggest Agent2 of being superior over Agent1.

Figure 15: Agent1 with risk assessment and move ordering enabled and Agent2 with no features enabled, showing error bars with a 0.999 confidence interval

Figure 16: Agent1 with risk assessment and move ordering enabled and Agent2 with only risk assessment enabled, showing error bars with a 0.999 confidence interval

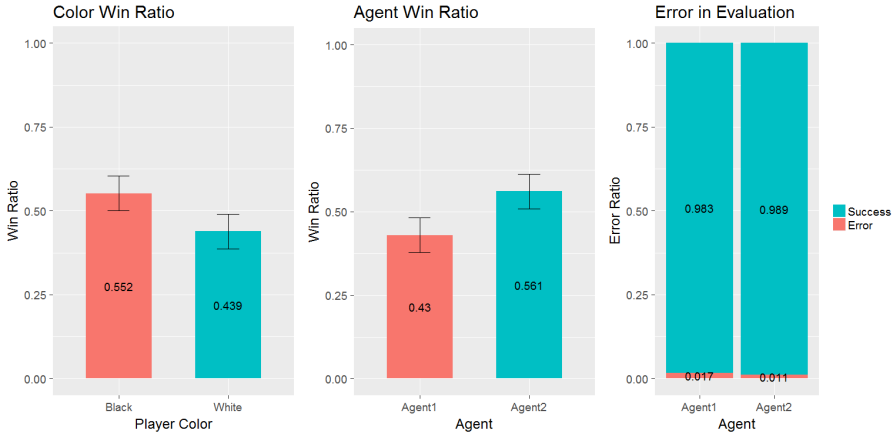


Figure 17: Agent1 with risk assessment and move ordering enabled and Agent2 with only move ordering enabled, showing error bars with a 0.999 confidence interval

Both results from the risk severity tests are displayed in Figure 18 and Figure 19. All Chi-square p-values show 2.2×10^{-16}, apart from win-ratio and agent in the first test, returning 1.5×10^{-10}. All tests strongly indicate dependent variables as H0 gets rejected. The LOS value indicate that Agent2 is superior over Agent1 with the values 0.0 and 1.0×10^{-6}

Figure 18: Agent1 with only risk assessment and Agent2 with risk assessment with a severity of 0.5. Error bars is displayed with a 0.999 confidence interval

Figure 19: Agent1 with only risk assessment and Agent2 with risk assessment with a severity of 0.1. Error bars is displayed with a 0.999 confidence interval

In the depth test (displayed in Figure 20), the Chi-square p-values for both color win-ratio and agent win-ratio show 1.9×10^{-9} and 1.8×10^{-8} respectfully. This means H0 gets rejected. This is different when compared to Figure 11 where both agents have only move ordering enabled. However, it is the agent who searches with a depth of 2 which gives a LOS value of 0.99.

Figure 20: Both agents with only move ordering enabled, but with Agent1 searching at a depth of 4. Error bars is displayed with a 0.999 confidence interval

6.1.2 Regression

The first regression test is looking at two identical agents, both having no features enabled. This is shown in Figure 21, and note it has been zoomed in on the y-axis. This has been done in order to make the different regression lines observable. Also, note that there are two lines for each color, these indicate the two different agents. This is also why the two groups of lines are symmetrical. Since both agents are equal, it is expected that the rating would not change as much. However, this is also related to the K-factor described in Chapter 3. For all tests, this was arbitrarily set to 20 which is the maximum amount of points one of the agents can take from the opponent in one game. In the graph, the faded line shows the actual rating after each game. This goes multiple times across ± 50 from its initial rating of 500, which indicates that an agent consecutively wins or loses. The thick faded line shows the confidence interval of 0.999 around the *moving average*. Since all three methods overlap, the other confidence intervals were left out, making the lines themselves stand more out. Since all three methods show close to identical trends, arguing for which is more accurate becomes difficult. Each line shows most difference in the beginning and end of the test, and judging by the beginning, where the rating is known, it is the GAM method which deviates most from approximating 500. The main trend of these two group of lines shows that they are continuously crossing each other.

In Figure 22 no lines starts to cross. Here Agent1 has enabled risk assessment while Agent2 has no features enabled. There is a clear difference in their strength, which can be shown in the corresponding test for independence in Figure 12. The lines become even more separated in the beginning compared to the last test. There is also a change in which line more accurately approximate 500, being the GAM method. The mean rating of the more superior agent is 551, this is obtained after playing around 130 games, from which the line starts to stabilize. The next test show very similar trends.

Figure 23 shows 2 very distinct agents. Here Agent1 has enabled move ordering while Agent2 has enabled risk assessment. Comparing against the last test, this does not show any lines crossing, which indicate a superior agent. This is also shown in the corresponding independence test (Figure 14). Looking at the beginning, each line is even more separable, making GAM the one which approximates the real initial value more closely. The mean of the superior agent, show a mean of 663 in rating, this is obtained after around 130 games have been played. From that point on, the trend seems to stabilize.

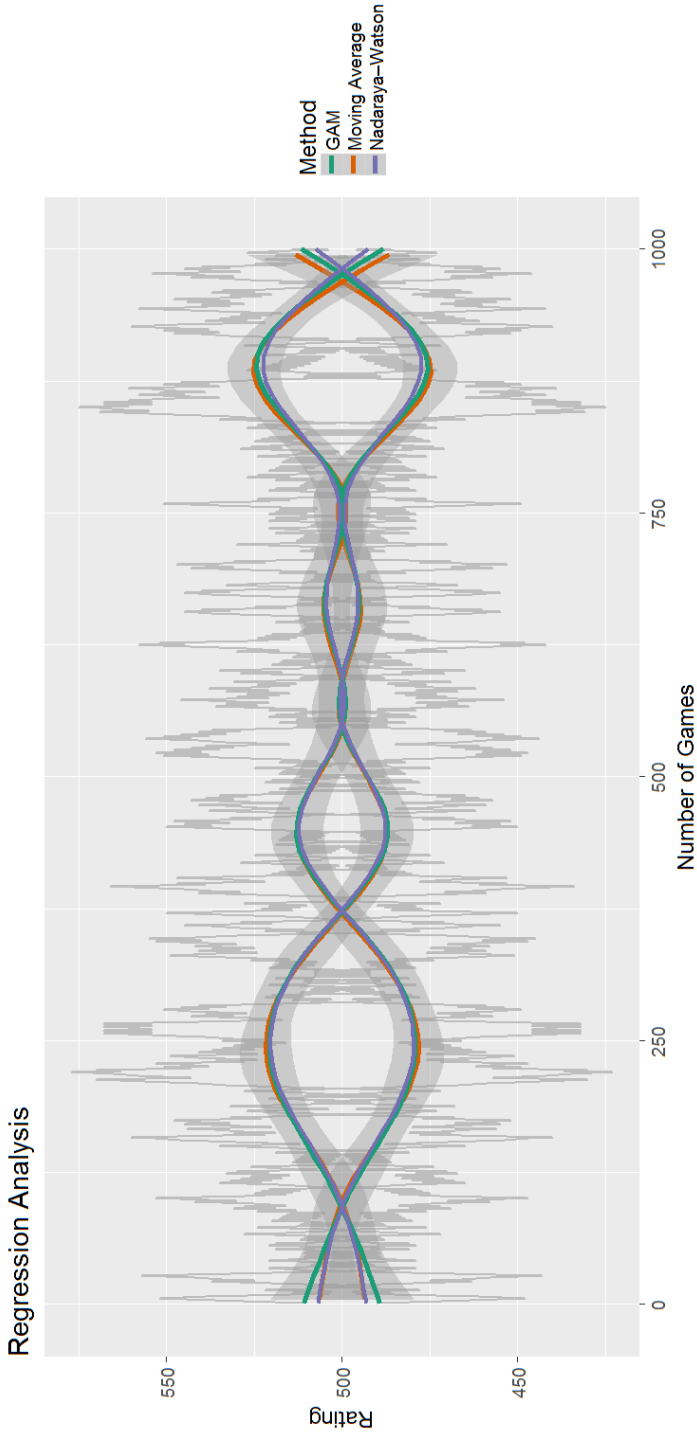


Figure 21: Show 3 regression lines, one for each agent. Both agents are equal with no features set

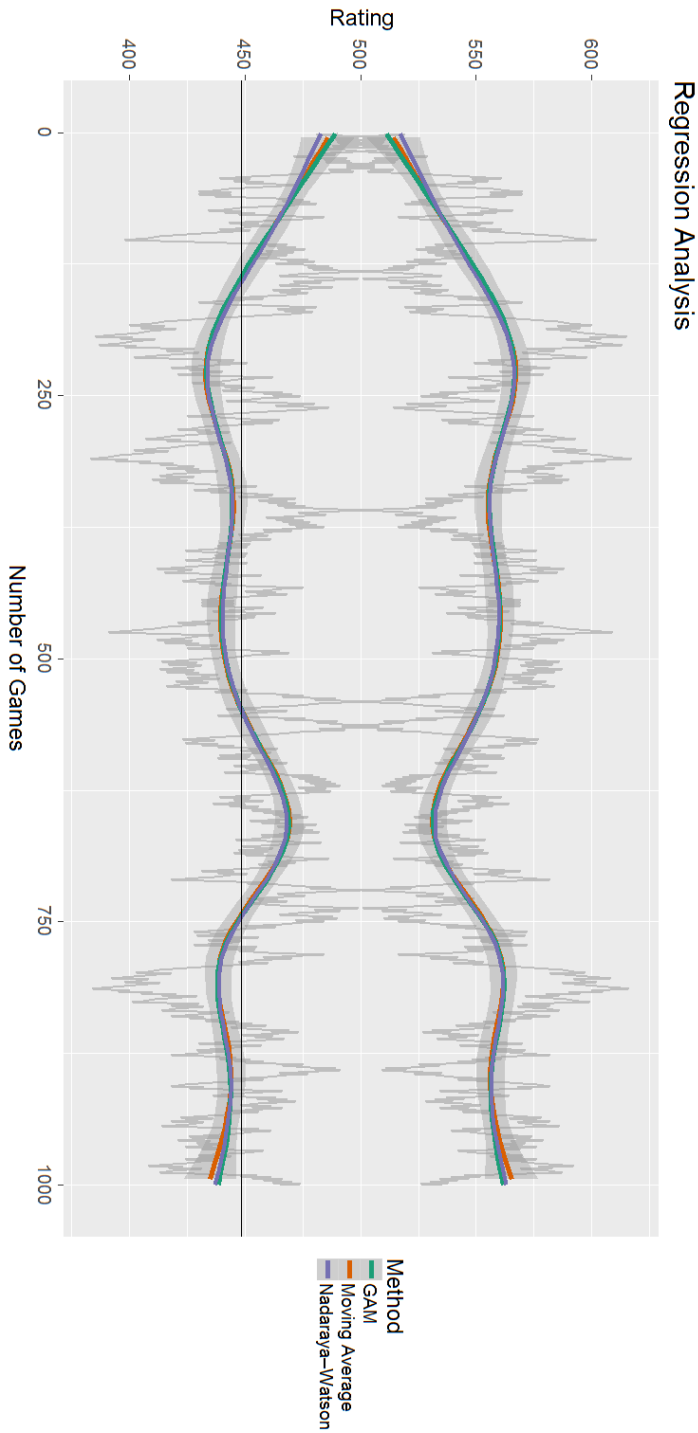

Regression Analysis
Rating
600
550
500
450
400
0
250
500
750
1000
Number of Games
Method
GAM
Moving Average
Nadaraya–Watson

Figure 22:

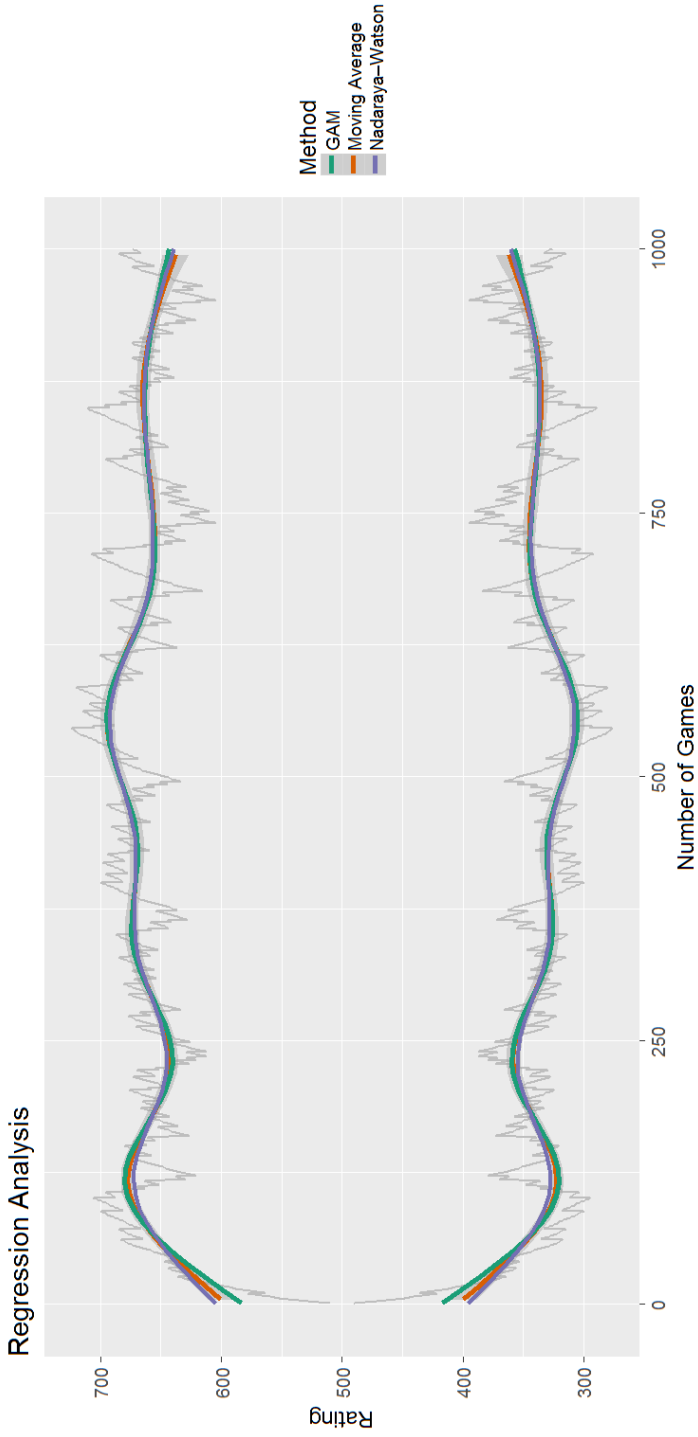


Figure 23: Show 3 regression lines, one for each agent. The better agent has only enabled move searching, while the worse agent has only enabled risk assessment

6.2 User Testing

6.2.1 Participants

Based on the answers from the 6 participants in the pre-questionnaire, the group was familiar with the rules of chess. However, their understanding of the game leaned more towards novice level. This is based on the fact that 3 of 6 have not played any traditional chess during the last month before the user test, and none have played more than 10 hours. Not only that, but 3 of 6 also answer they did not know how to play positional chess at all, while 2 of 6 did not have any knowledge about chess openings. Nevertheless, 2 participants seemed to be confident in their chess knowledge, selecting average or above when it came to chess openings, tactics, and playing positional chess.

In the post-questionnaire 5 of 6 believed the middle and end game were most challenging to play. The main arguments were that it became tougher to anticipate the moves done by the agent and to find good moves for themselves. Therefore, when asked which phase of the game was the easiest to play 5 of 6 answered the opening. Before making a move the participants considered between 2 and 4 alternatives on average. When considering moves and responses by the agent, 3 of 6 answered that they only considered the move to make. The deepest a participant thought were 4 consecutive moves before deciding what to play. When it came to how risky they played on average, all participants answered average or above. One stated selecting only risky moves, while another did more risky moves then safe.

6.2.2 Games

After going through all 12 games, move by move, the Assumption 4.4 provided in Chapter 4, seem to be correct. In every position where the participant could capture a higher valued piece, they did in fact, capture it. Dark pins did not occur for either the participants or the agent in any of the games. However, one participant manage to be put in checkmate. This is shown in Figure 24. The agent has captured the dark-squared bishop meaning it is not afraid of a bishop being on a1, b2, or c3, which could potentially have lost the queen when moving the rook. Instead, the king is revealed in a position where the participant knows it is in checkmate.

Another game the agent manage to make a discovered attack with a separate attack, resulting in winning the game. This is illustrated in Figure 25, where the agent play with white pieces, and moves the bishop to c4, making a discovered threat against the opponent's queen on d8, but also attacking the king on e6. Note that the bishop is not observable for black after the move. The participant decided to save the queen by moving it to a safer position. Unfortunately, it loses the king due to the hidden bishop on c4.

(a) Black to move

(b) White is in checkmate

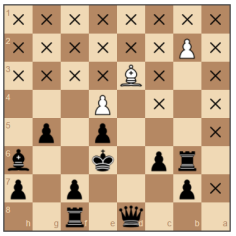
(c) Black to move

(d) White is in checkmate

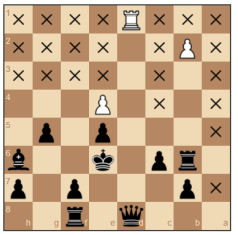
Figure 24: Check Mate: (a) and (b) shows it from white's perspective, before and after black moves its rook to d2, while (c) and (d) shows the same, but from black's perspective

(a) White to move

(b) Queen and king is under attack!



(c) White to move



(d) "Only" the queen is under attack

Figure 25: Double Attack: (a) and (b) show it from white's perspective, before and after the attack, while (c) and (d) shows the same, but from black's perspective

One of the mistakes the agent does is sacrificing bishops and knights for pawns on the second rank. This makes sense, as there is no risk of being captured by a pawn on the first rank as that is not possible. This leaves the knight being the only less valued piece for the bishop, but when both knights are visible the agent believes the capture will end being a pawn up. This happened in multiple games and was also mentioned by one of the participants, stating the agent became very aggressive around the middle game. In one game, the participant did not play positionally, meaning most pieces were lost because they were not protected. Overall, the agent managed to win 50% of the games, 3 by white and 3 by black. It should be noted that in one of the games the participant did not seem interested in winning as all pieces got lost, ending in moving the king to the center to be captured.

In one of the games, the participant seems to have a good position materially, after a couple of moves. Unfortunately, the queen gets lost because of a hidden knight. In the end, the participant loses the game because of an open file, providing with dangerous attacking opportunities with a rook. After losing more material and

not considering possible threats to the king resulted in losing the game.

Another game shows that the agent is afraid of exposing the queen for rooks, which result in sacrificing a knight for a pawn. In a separate game, the agent manages to win a lot of material, leaving the participant with only 3 pawns and a king, where the agent is up 2 rooks and a bishop. The participant manages to win by getting a pawn promoted to a queen. The agent could have stopped the promotion, but did not because of a potential risk of being captured by another pawn, both if capturing on c7 or by stopping it, moving to c8. This is illustrated in Figure 26.

In two of the games, the agent gets in repeat mode, meaning it moves the same piece back and forth. This is usually the rook or the queen, and it happens because other moves might expose them. This, in turn, prevent the agent in castling, which make these scenarios a weakness for the agent.

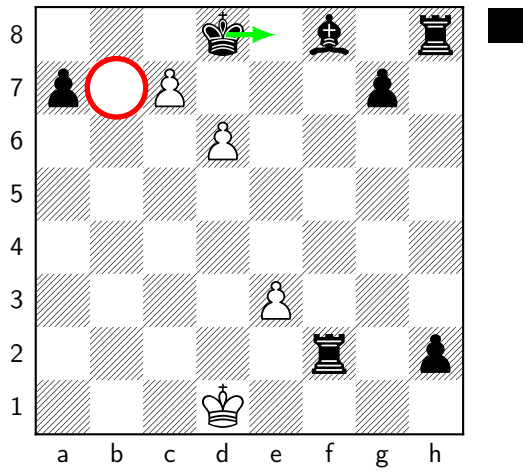


Figure 26: Agent allows pawn promotion since b7 is unknown and therefore might contain an enemy pawn, f7 is however visible due to the rook on f2

7 Discussion

7.1 User Tests

Seeing how the agent behaves against people have gained more insight of its weaknesses and strength. Even though the participant group was relatively small, each played 2 games resulting in a total of 12 different games that has been examined.

The fact that the agent manages to win 50% of the time both as white and black should not be taken too much into consideration. The games themselves confirm their chess knowledge as covered in the pre-questionnaire of being at a novice level, which is also backed up by observing a difference in intentions over the board. Most participants played above average risky moves, where one, in particular, did not seem interested in winning, as all pieces became sacrificed from the opening. Hammer pointed out that Darkchess is more about playing positional than taking chances, and the goal in the opening is very similar to traditional chess. Getting pieces developed while taking control of the center, but also play prophylactic moves, prevent the opponent from utilizing possible weaknesses by playing defensive. He also pointed out that it is more important to keep one's own king safe then locating and attacking the opponent's king. Unfortunately, half of the participants had no knowledge about playing positional chess.

Nonetheless, based on the games the agent performs reasonable moves in the opening, getting pieces developed and starts taking control over the center. However, the agent does not consider moving pieces to unprotected squares, which results in sacrificing bishops and knights for pawns. This start to occur after the opening, where the agent starts attacking pawns on the second rank. Another reason could also be that the risk is low for losing a knight or bishop for a lower valued piece on the second rank. This makes sense as it is not possible to have pawns on the first rank.

The main weakness of the agent is when the agent starts repeating moves. This occurs when the agent itself is down in material, and become too afraid of exposing the pieces it considers to move. There could be different reasons for this, first of all, the agent might consider the best move first which will result in only one move alternative. Another reason could be that among the move alternatives, one stands out, having a bigger chance of being selected. How risk assessment is implemented has also an impact on this, as the probabilities assume a uniform distribution. One could normalize the piece square tables, using it as a distribution for where pieces

are more likely to be placed.

Based on the independent tests, risk assessment alone show poor results compared to when no risk assessment is enabled. Knowing if this applies to people is unknown. However, based on the assumption (which the data indicate being true) suggest it is more likely. Apart from not observing any dark pins, double attack seems to be relatively dangerous, especially since it provided a victory for the agent. This suggests long attacks, which is provided by bishops and rooks, can provide tactical strategies. This also mean that the agent should exploit open diagonals and files. However, in order to accomplish that, the agent needs to obtain knowledge about the opponents pawn structure. This is also an important and well-known evaluation technique as it includes positional weakness.

7.2 AI Features

The fact that playing as white shows to be, in most cases, an advantage, keep in mind that both agents use the same evaluation function. This means that they evaluate each position in a similar manner, considering risk and material in the same way. This especially limits the opening, as long as no information is shared, both agents consider very similar moves. This makes sense though, as their goal at this point is to develop pieces to better squares, which help in taking control of the center.

In the games where no features are set, it seems that the default move ordering has an effect on the outcome. The default move ordering is different from white and black since move generating scans the board from a8 to h1 for both players. This makes the rook blacks first piece being checked for legal moves, but for white it becomes the a2 pawn. From the examined games in Appendix A. The first game illustrates this disadvantage for black as black are more likely to start repeating moves. White, on the other hand, gets its central pieces being selected first, getting a more varied move order. White seem to be more likely to attack as well, and when this affect the king, it gets more vulnerable when making sub-optimal moves.

This argumentation seems plausible for when risk is enabled as well. However, the risk assessment also affects the move selection when the king becomes too exposed. In these scenarios, the agent prioritizes moving the king back and forth in an attempt to save the king from possible threats, completely ignoring whether the opponent gets pawns promoted to queens.

When move ordering is enabled against no features, black is more likely to win. According to the examined games, this reason seems to be of prioritizing captures, making black recapture whenever white attacks. This result in white ending up with fewer pieces and an exposed king, making it more likely for black to capture it.

The reason why no risk seems to be a better strategy then applied risk is because recapturing pieces might oppose certain risk, for which it gets prevented. In other words, the agent which does not consider risk acts more aggressively and starts exchanging pieces for less valued pieces. The opponent can become too afraid to recapture as it might expose other more valuable pieces. This has been indicated in numerous tests, where scaling the severity of risk assessment from 0.5 to 0.1 only approximate the results of not assessing risk at all. This becomes clear when comparing Figure 18 and 19 against Figure 12. The main reason behind this weakness is that the agent does not consider opponent pieces being unprotected. As related work suggest, move ordering improves the evaluation function in terms of selecting better moves first, which in turn also optimizes searching as more alpha-beta cutoffs occur, limiting the search on unnecessary moves.

With added risk assessment, the first to win material reduces the risk considerably, especially when capturing lower valued pieces, such as bishops or knights. Since a bishop is restricted to move on a specific colored square, capturing one makes all moves to the same colored square reduce in risk. This was shown in the checkmate scenario, displayed in Figure 24, where no risk is applied for moving the rook, which "exposes" the queen on the diagonal.

When it comes to known evaluation methods in traditional chess, the material has been shown to be manageable as it has been implemented to a working darkchess agent, only slightly being affected by the applied uncertainty. For positional weakness, more knowledge is required about the opponents pawn structure, making it possible to reason for how many double, backward, or isolated pawns the opponent have. The uncertainty makes obtaining this information non-trivial due to adverse surprise moves as displayed in Figure 3. Even though the agent should avoid letting the opponent perform these moves, it needs to have a strategy nonetheless. For instance, it should be safe to exclude the king, especially the opposite colored bishop. Further, it is the observable space which determines other possibilities such as possible knight attacks. Because of this special move, which one would like to avoid, prevents knowing exactly when pawns change files.

Interestingly, without these moves, the agent can more easily reason about the opponents pawn structure, along with more accurately calculating the risk for pawn threats. Since pawns cannot move backward, observing the initial square for pawns will reveal useful information. Observing for instance that the pawn has moved will limit the number of possible dark squares it can be positioned in (regardless whether you lose sight of the square), simply because it is approaching the visible space. Unfortunately, because of adverse surprise moves, knowing whether a pawn has captured an unprotected piece makes it more challenging as there could be two pawns in a file. An alternative could be to calculate risk in this manner then

fall back with the current solution when an adverse surprise move occur where the attacking piece could potentially be a pawn.

Mobility is also difficult to obtain in darkchess. This relates of course to how many pieces the opponent have, meaning it is possible to argue that a player with more pieces also are the player with more move options. However, there are exceptions if one have blocked pieces such as the rooks, bishops, queen and king in the initial position.

That being said, uncertainty affects traditionally evaluation methods differently, but nonetheless, makes it more challenging to reason about what the real state is. However, note that by addressing the pawn structure problem one can more accurately reason about pawn threats but also other evaluation methods such as king safety. These are related as the pawns in front of the king should not move as it could expose it to threats.

Overall, most results show there is a clear difference between agents. This is both indicated by independent tests and the likely hood of superiority. However, relying on the LOS value alone can provide width inaccurate claims as it suggests one of the equal agents are superior. This indicates the provided LOS equation is very sensitive to game results.

The regression tests show clearly whether an agent is more superior. Comparing the 3 regression methods, it is difficult to distinguish between them. This seems to strongly indicate a clear trend, and knowing exactly which are more accurate does not become as important. From the last two tests, both indicate that if there is any difference, it becomes clear after about 130 games.

7.3 Evaluation & Belief State

In darkchess, one has full knowledge about material on the board. This information has been utilized and is used when evaluating any position. Unfortunately, the use of piece square tables provided with challenges as the exact position of each opponent piece are unknown. The implementation assumes that each piece is equally likely to be positioned in any square, given it is legal. This is, of course, inaccurate to assume, first of all this approach does not consider the number of moves that has been made from the opening. For instance, only after 1 move, a knight is not possible to be located on every square. This includes all pieces, for instance, both rooks are blocked, meaning it requires multiple moves before one can assume a rook can be positioned in any squares within the unobservable space. Secondly, piece placement also depends on different factors such as a player's current goal, and current knowledge of the opponent. The former might be playing positional or aggressively if the position allows both, while the latter highly depend on the move history of a game. This argument, however, depends on who is playing, making it difficult to

observe any general distribution. Nonetheless, the results of the user testing suggest a uniform distribution provided with relatively challenging play against novice players. An alternative approach could be to normalize each piece square table and used it as a distribution for where pieces are more likely to be placed. This assumes that the opponent is more likely to place pieces on good squares.

Piece square tables provide the agent to distinguish between better squares for each piece type. According to related work, this provides with *semi-decent* chess play, which the outcome of the user tests confirm, even when only evaluating material and avoiding losing quality.

With this approach, the belief-state is basically numbers for how many of each piece type is visible and hidden. One might think these values are always known and to be 100% accurate. However, because of promotions (and only promotion), material can change without one knowing. On another note, it is more specifically the hidden number which represent the belief state. When risk assessment included, the belief state in total becomes the combination of a uniform distribution of the dark space with the hidden piece type numbers. With this implementation and with the search performed only within the visible space, alpha-beta search becomes possible. The related work, provide with numerous alternative for search methods, although alpha-beta seem to be the popular approach, especially in chess. For partially observable chess domains, both MCTS and metaposition have shown applicable solutions where the latter showed more promising results. The belief state can also be a distribution over all possible states which is the case when following a POMDP approach. How exactly this is adopted to a chess domain is unknown, as most examples where POMDP has been applied, actions are stochastic, meaning the exact outcome of an action is not known in advance. POMDP is also claimed of being most relevant for relatively small problems. Pseudo pieces in metaposition can be viewed as a probability distribution for where one can expect certain piece types on which squares. This approach also takes move order into account. Based on this, POMDP can be appropriate.

7.4 Search

Increasing the search depth for one agent showed that it influenced the game results negatively. One of the factors here is the search is limited to only consider visible moves for the opponent. Meaning, the agent does not consider moving an opponent piece to the unobservable space. In fact, if there is only one visible piece by the opponent, the agent does not consider this piece to *not* move. This can result in considering inaccurate responses, and note that this applies to search with a depth of 2 or more. Take for instance the position in Figure 27. Here, when the agent considers moving away from the knight on c6, the only responses by the opponent (given that it has to move) result in a worse position. In other words, every respond loses the knight immediately. This indicates that preventing the opponent to capture the knight on c6 will lead to a better position. In reality, the knight is strongest in the center, and thus, as long as there are unobservable alternative moves, not moving any visible pieces should also be considered.

Figure 27: A dangerous knight peeking into black's position

As mentioned in Chapter 4, the risk is only applied to the first move the agent considers. If risk assessment has been implemented similar to how evaluation works, the risk would only have been applied to the deepest state. Imagine the agent is starting to search from the position displayed in Figure 27, where the first move it considers is knight to b4. This exposes the king from a potential bishop or queen on a4. White's best responds (when it has to move the knight) becomes knight captures pawn on f7 which gets lost after king recaptures. Now the desired depth has been reached, and assessing the risk in this position would indicate the king being completely safe on its new square. The drawback is that this method does not account the risk of the first move. Ideally, risk should be applied after each agent move and passed down as the search goes deeper. How the risk from agents first move should be incorporated with its second move (a depth of at least 3) has not been discussed as the focus of this research have mainly been on search with depth 2. Nonetheless, this is why risk assessment is not performed in quiesce search.

Even though with a simplified evaluation function, alpha-beta seems be work relatively reasonable with applied risk to deal with the uncertainty. That relates to searching with a depth of 2. Although, searching deeper requires enhancements such as move ordering, the relevance of applied alpha-beta search at a deeper level is unknown and needs further investigation.

According to user tests, move searching seem to be better in the openings then middle and end game. As one participant commented, the agent becomes very aggressive from the middle game as it starts sacrificing bishop and knights. User tests have also shown that the agent evaluates threats inaccurate in the end game because of the risk assessment. This suggests that the evaluation becomes less stable from the middle game. This indicate that different evaluation strategies could be selected depending on the phase of the game.

According to the user tests, searching at a depth of 2 might suggest that deeper move search is not that relevant according to novice players. One possible explanation could be that the middle and end game phase were more challenging as is it were more difficult to reason about what moves the agent performed. As such, more considerations has to be made for each depth, making it more demanding to think deeper. Knowing how relevant this applies for professional chess players is unknown.

7.5 Limitations

For gathering participants for the user tests, Gjøvik Sjakklubb were contacted on Facebook. Unfortunately, their respond was too late in order to gather experienced chess players. Contacting the club could have been done earlier, having scheduled the user tests. However, getting a chess agent play reasonable were higher prioritized at the time, fixing decisive bugs.

8 Conclusion

Darkchess has been an untouched domain regarding artificial intelligence and reasoning under uncertainty. It has been obtained a considerable amount of knowledge and an insight into the challenges of darkchess, along with gathered an experts opinion on the matter. In addition, a working as agent has been developed, able to play darkchess seemingly at a novice level according to user testing.

Multiple tests have also been conducted between different agents, able to observe if any agent is more likely superior over another. It has mainly been focused on risk assessment and move ordering, but searching with different depth has also been tested. The results are discussed with respect to the applied belief state, evaluation methods, risk assessment and search algorithm. This includes how stable the applied evaluation function is over the duration of a game.

It has been shown that searching only within the visible space of a partially observable environment, where actions can affect the observability can achieve an agent leaning towards semi-decent playing strength. The main factors for this include the use of piece square tables, a simplified evaluation method which only includes material balance. This also includes assessing the risk to prevent worst case actions. Risk assessment has shown to have relatively small impact on the decision process compared to move ordering. The results also suggest that the implemented solution perform best in the opening phase of the game from which it gets too aggressive. However, from that point, it also becomes more challenging for people to find good moves which again has affected the game results. The challenges in applying traditional evaluation methods have shown different impacts, where material evaluation has been successfully applied. That being said, the uncertainty in darkchess affects the decision maker depends on the material and risk assessment. This includes the position of the king and how exposed it it as well. In short, it becomes about taking chances preventing a worse position, or accepting a known but slightly worse position.

When considering the performance of the evaluation function over a duration of a game, it has been shown to start sacrificing pieces, thus selecting moves which weaken its position. At the same time, risk assessment in certain cases affects the evaluation at a level which obscure other move alternatives.

Apart from a working darkchess agent, this research has contributed to opening up a new domain for investigating reasoning under uncertainty. With it, a modified alpha-beta pruning algorithm is provided in order to make search possible in a

partially observable environment, which can be applied to other similar domains. The developed solution is also open sourced using node.js, not limiting any OS platforms, making further work possible for any researcher.

By further investigating this field of research will provide the AI community with more knowledge on different approaches dealing with uncertainty, specifically in complex domains such as darkchess, and more generally towards 2 player turn-based strategy games.

8.1 Further Work

In order to pursue the approach covered in this thesis, a natural next step would be to investigate how to apply more evaluation methods, such as positional weakness and mobility. Another natural next step would be to investigating whether normalized piece-square tables could replace the uniform distribution in risk assessment. These have been briefly mentioned.

Other further work, with improved search flow, could investigate search improvements, allowing for deeper search with applied risk. With assessing pawn structure, one can also start applying other evaluation methods such as *king safety*. One could argue that by observing the opponent's pawn structure, one can infer where the opponents king is positioned since moving pawns in front of the king exposes it to threats. Other possibilities are looking into different search algorithms such as MCTS, or apply different techniques such as metapositions, or looking into POMDP as a possible approach.

Bibliography

[1] Russell, S. J. & Norvig, P. 2010. *Artificial Intelligence Third Edition*. Pearson.

[2] Poole, D. & Mackworth, A. Artificial intelligence - foundations of computational agents – 6 reasoning under uncertainty (online). 2017. URL: http://artint.info/html/ArtInt_138.html (Visited February 2017).

[3] Hall, H. Uncertainty in medicine « science-based medicine (online). 2016. URL: https://www.sciencebasedmedicine.org/uncertainty-in-medicine/ (Visited December 2016).

[4] Nielsen, J. Darkness chess (online). 2017. URL: http://www.chessvariants.com/incinf.dir/darkness.html (Visited February 2017).

[5] Wikipedia. Dark chess - wikipedia (online). 2017. URL: https://en.wikipedia.org/w/index.php?title=Dark_chess&oldid=762632327 (Visited February 2017).

[6] FIDE. Fide counrty top chess players (online). 2017. URL: https://ratings.fide.com/topfed.phtml?tops=0&ina=1&country=NOR (Visited April 2017).

[7] Shannon, C. E. 1950. Programming a computer for playing chess. *Philosophical Magazine*, 41(314), 18. URL: http://archive.computerhistory.org/projects/chess/related_materials/text/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon.062303002.pdf (Visited February 2017).

[8] Favini, G.-P. *The dark side of the board: advances in chess Kriegspiel*. PhD thesis, alma, 2010.

[9] Wikipedia. Shannon number - wikipedia (online). September 2016. URL: https://en.wikipedia.org/w/index.php?title=Shannon_number&oldid=739682507 (Visited February 2017).

[10] Wikipedia. Exptime - wikipedia (online). October 2016. URL: https://en.wikipedia.org/w/index.php?title=EXPTIME&oldid=742310238 (Visited February 2017).

[11] Yen, S.-J., Chou, C.-W., Chen, J.-C., Wu, I.-C., & Kao, K.-Y. 2015. Design and implementation of chinese dark chess programs. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1), 66–74.

[12] Jouandeau, N. & Cazenave, T. 2014. Small and large mcts playouts applied to chinese dark chess stochastic game. *Computer Games*, 504, 78–89.

[13] Fernández, A. & Salmerón, A. 2008. Bayeschess: A computer chess program based on bayesian networks. *Pattern Recognition Letters*, 29(8), 1154–1159.

[14] Junghanns, A. 1998. Are there practical alternatives to alpha-beta in computer chess. *ICCA J*, 21(1), 14–32.

[15] Wiki, C. P. chessprogramming - alpha-beta (online). 2017. URL: https://chessprogramming.wikispaces.com/page/diff/Alpha-Beta/604483933 (Visited May 2017).

[16] Mannen, H. 2003. Learning to play chess using reinforcement learning with database games. *CKI Scriptieserie*.

[17] Wiki, C. P. chessprogramming - iterative deepening (online). 2016. URL: https://chessprogramming.wikispaces.com/page/diff/Iterative+Deepening/601765644 (Visited May 2017).

[18] Kościuk, K. & Drużdżel, M. 2010. Player modeling using bayesian networks. *Symulacja w Badaniach i Rozwoju*, 1(2), 151–158.

[19] Stockfish. Stockfish/search.cpp at 732aa34e3dec39de9c80a07f6ecba7cb0569b95e · official-stockfish/stockfish (online). 2017. URL: https://github.com/official-stockfish/Stockfish/blob/732aa34e3dec39de9c80a07f6ecba7cb0569b95e/src/search.cpp (Visited May 2017).

[20] Wiki, C. P. chessprogramming - quiesce search (online). 2017. URL: https://chessprogramming.wikispaces.com/page/diff/Quiescence+Search/609935281 (Visited May 2017).

[21] Russell, S. J. & Norvig, P. 2010. *Artificial Intelligence Third Edition*. Pearson.

[22] Rodriguez, A. C., Parr, R., & Koller, D. 1999. Reinforcement learning using approximate belief states. *Advances in Neural Information Processing Systems*, 1036–1042.

[23] Wiki, C. P. chessprogramming - point value (online). 2016. URL: https://chessprogramming.wikispaces.com/page/diff/Point+Value/584327187 (Visited May 2017).

[24] Wiki, C. P. chessprogramming - piece-square tables (online). 2016. URL: https://chessprogramming.wikispaces.com/page/diff/Piece-Square+Tables/600433626 (Visited May 2017).

[25] Michniewski, T. 1995. Samouczenie programów szachowych. *Praca magisterska, Wydział*.

[26] Glickman, M. E. & Jones, A. C. 1999. Rating the chess rating system. *Chance*, 12(2), 21–28.

[27] FIDE. Fide chess rating calculators: Chess rating change calculator (online). 2017. URL: https://ratings.fide.com/calculator_rtd.phtml (Visited May 2017).

[28] Bester, D. & von Maltitz, M. 2013. Introducing momentum to the elo rating system. *University of the Free State: Department of Mathematical Statistics and Actuarial Science*.

[29] team, C. Ccrl 40/40 - index (online). 2017. URL: http://www.computerchess.org.uk/ccrl/4040/ (Visited May 2017).

[30] Wiki, C. P. chessprogramming - simplified evaluation function (online). 2014. URL: https://chessprogramming.wikispaces.com/page/diff/Simplified+evaluation+function/533490220 (Visited May 2017).

[31] Wiki, C. P. chessprogramming - 0x88 (online). 2016. URL: https://chessprogramming.wikispaces.com/page/diff/0x88/600596996 (Visited May 2017).

[32] Wiki, C. P. chessprogramming - negamax (online). 2015. URL: https://chessprogramming.wikispaces.com/page/diff/Negamax/559016615 (Visited May 2017).

A Agent vs Agent Games

A.1 No Features

1 ♘c3 ♘c6 2 ♘f3 ♘f6 3 d4 d5 4 e3 ♗e6 5 ♘b5 ♕d7 6 ♘×a7 ♘×d4 7 ♘b5 ♘×f3 8 ♘×c7 ♕×c7 9 g×f3 ♖d8 10 ♗d3 ♔d7 11 ♗e4 ♘×e4 12 ♕×d5 ♔c8 13 ♕×d8 ♔×d8 14 f×e4 ♕×h2 15 ♗d2 ♕g3 16 f×g3 ♔e8 17 ♗c3 ♔d7 18 ♗×g7 ♗×g7 19 O-O-O ♖e8 20 ♖×d7

1 ♘c3 ♘c6 2 ♘b5 ♘h6 3 d3 d6 4 ♗×h6 e5 5 ♗×g7 ♗d7 6 ♗×f8 ♔×f8 7 ♘f3 ♕c8 8 ♘×d6 c×d6 9 ♘×e5 ♘×e5 10 e4 ♕c4 11 d×c4 ♖d8 12 ♕h5 ♔g8 13 ♕×h7 ♔×h7 14 h3 ♗×h3 15 ♖×h3 ♔g8 16 ♖×h8 ♖e8 17 ♖×g8

A.2 Risk Assessment

1 ♘c3 ♘c6 2 d4 ♘f6 3 ♗d2 d5 4 a3 e6 5 ♘f3 ♗d6 6 a4 O-O 7 e3 ♗d7 8 ♗e2 ♕e7 9 ♗d3 ♖ad8 10 O-O ♖fe8 11 ♗×h7 ♖f8 12 ♗×g8

1 ♘c3 ♘c6 2 d4 ♘h6 3 f3 d5 4 ♗×h6 ♗d7 5 ♗×g7 ♗×g7 6 f4 O-O 7 f5 ♖e8 8 f6 e×f6 9 ♘f3 ♕e7 10 e3 ♕d6 11 e4 d×e4 12 ♕e2 ♘e7 13 ♘d2 ♗e6 14 O-O-O ♖f8 15 ♘cb1 f5 16 ♕e1 ♖ad8 17 d5 ♖c8 18 d×e6 f6 19 a3 ♕e5 20 h3 ♖cd8 21 a4 ♖×d2 22 ♖×d2 ♖d8 23 ♗c4 ♖×d2 24 ♘×d2 ♗f8 25 ♔b1 ♔h8 26 ♘b3 a6 27 a5 b5 28 ♗f1 ♘d5 29 ♘d2 ♔g7 30 e7 ♔g8 31 e8♕ ♔h8 32 ♕×f8 ♕e8 33 ♕×h8

A.3 Move ordering vs No Features

1 ♘c3 d6 2 ♘f3 ♘f6 3 ♘d5 ♘×d5 4 d4 ♘c6 5 a3 ♘×d4 6 ♘×d4 ♗e6 7 e3 ♘×e3 8 ♗×e3 d5 9 ♘×e6 f×e6 10 ♗×a7 ♖×a7 11 ♗e2 e5 12 O-O e6 13 ♗b5 c6 14 ♗×c6 b×c6 15 ♕×d5 c×d5 16 h3 ♕d6 17 ♖ad1 ♖×a3 18 b×a3 ♕c6 19 ♖×d5 e×d5 20 ♖e1 ♗×a3 21 f3 O-O 22 c4 ♕×c4 23 ♖f1 ♕×f1 24 ♔×f1 d4 25 f4 ♖×f4 26 h4 ♖×f1

1 ♘c3 ♘c6 2 d4 ♘×d4 3 ♕×d4 ♘f6 4 ♕×f6 e×f6 5 ♘d5 ♗d6 6 e4 O-O 7 ♘×c7 ♕×c7 8 ♘f3 ♕×c2 9 ♗e3 ♕×b2 10 O-O-O ♕×a2 11 ♖×d6 ♖d8 12 ♖×d7 ♗×d7 13 ♘d4 ♗e6 14 ♘×e6 f×e6 15 ♔b1 ♕×b1