3D CHESS GAME ENGINE

Computer Based Game

ABSTRACT

FIRST AVAILABLE GAME FOR 3D CHESS WITH A 3D CHESS AI, ENVIRONMENT AND UI. IT WILL ALSO HAVE CAPABILITIES TO HAVE TWO PLAYERS OVER A WAN AND BASIC USE OF DATABASES.

THOMAS BALE

A-LEVEL PROJECT

TABLE OF CONTENTS

1.0 ANALYSIS

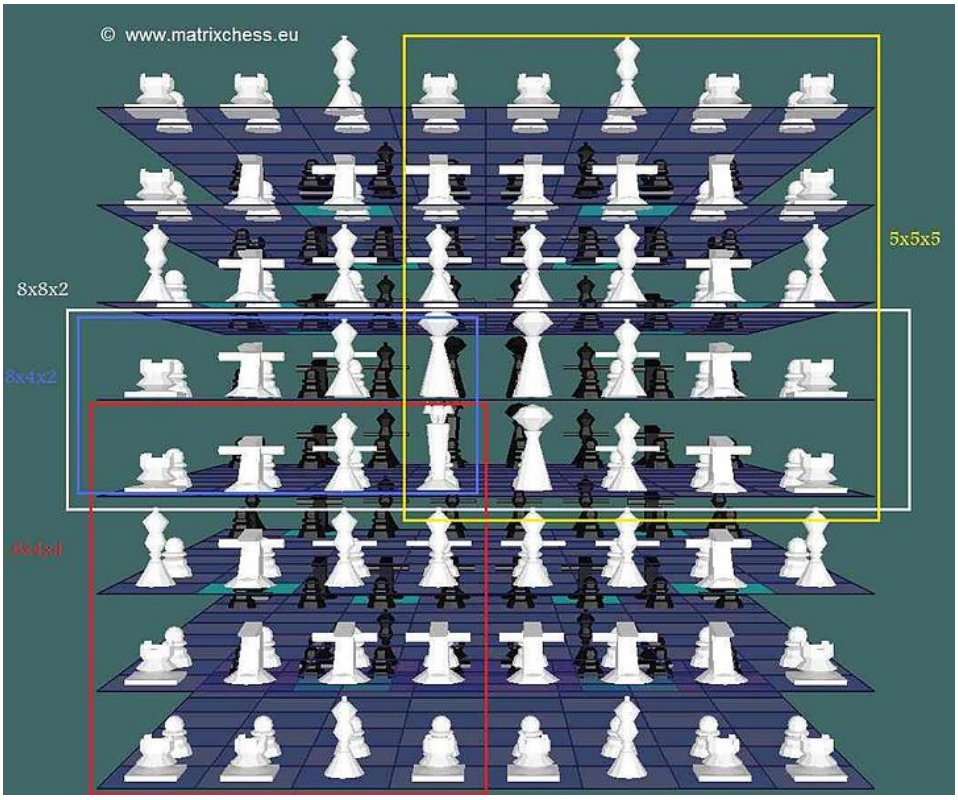
1.1.1 – PROJECT PROPOSAL – PROJECT OUTLINE

Oscar, an intelligent student at sixth form, is an avid chess player who likes to play many different chess variants. On top of this, he also enjoys playing chess on his computer and mobile device. His favourite variant is 'Strada chess' in which the chess pieces move through three stacked boards. What he enjoys most is the 3D aspect of the game as it allows a for a higher for in depth level of thinking.

There lie two problems for Oscar: he would like to play another version of chess where the game is more 3-dimensionally involved, whilst also being able to play the game without physical pieces or setup. Therefore, Oscar would like to have a game produced where he is able to play on an electronic device where the game works in the same degree as chess, but with altered rules.

As this would be a new game for a niche group other than Oscar he would also like to play against an AI as a real opponent may not be available to play when he would like. This AI will have to be tailor made to this specific game as a standard chess AI will not compute the necessary moves.

1.2.1 – BACKGROUND – WHAT IS 3D CHESS?



(Note – FIDE chess is the tournament standard 'normal' chess)

3D chess is played inside an 8x8x8 cube as opposed to the FIDE chess 8x8 board. One set of 32 pieces is used per layer (totalling to 256 pieces). The pieces also move differently from FIDE chess to allow them to move between 'layers'. A representation can be seen above:

1.2.2 - BACKGROUND – RULES OF 3D CHESS

To fully understand how 3D chess works the rules that differ from FIDE chess are shown below (rule of FIDE chess at https://www.fide.com/FIDE/handbook/LawsOfChess.pdf).

- The playing field is an 8x8x8 cube instead of an 8x8 square.
- All squares in a single vertical stack are of the same colour: for example, a1 is a black square on each of the 8 8x8 levels.
- The standard algebraic notation is modified by the addition of the level number which is prefixed to the name of each square. Thus, the bottommost square a1 is 1a1, and a1 on the top level is 8a1.
- Eight standard chess sets are used. Each player has the standard Chess line-up on each of the eight levels.
- The Kings on 4e1 and 4e8 must be distinguished from the others in some obvious way. Rules about check and checkmate apply to only these two Kings; the other Kings are not royal, and in fact are called Commoners.
- Castling can involve Kings and Rooks from different levels.
    - It is obvious that from 4e1 the White King can Castle to 4g1 bringing the Rook from 4h1 to 4f1; after all, this is all on the same level, so it looks like FIDE Chess.
    - Equally obvious, "from 4e1 to 2g1 with 1h1", "from 4e1 to 6g1 with 7h1", "from 4e1 to 4c1 with 4a1", and "from 4e1 to 6c1 with 8a1".
    - However, "from 4e1 to 2c1 with 1a1" is a special rule: it really ought to be "with 0a1" but there is no level 0.
    - Vertical Castling, for example "from 4e1 to 6e1 with 8e1", is not allowed.
    - The notation for Castling "from 4e1 to 2g1 with 1h1" is "O-O(2g1)", in other words, we give the King's destination position as part of the move; this can be omitted if only one O-O is possible.
- The normal two-dimensional moves of the pieces are translated into three dimensions in the following manner:
    - A Rook makes a standard Rook move, or moves straight up and down, or rises or descends as it makes a Rook-move. For example, a Rook on 1a1 can (if nothing is in the way) move to 8a8, passing through 2a2, 3a3, 4a4, 5a5, 6a6, and 7a7.

- A Bishop makes a standard Bishop move, or rises or descends; for example, from 1a1 to 8h8 via 2b2, 3c3, and so on.

- The Queen combines the powers of the Rook and Bishop (as usual).

- The King and the non-royal King which we call the Commoner move as the Queen, but just one square at a time.

- Pawns rules are as follows:
  - Pawns may only move straight forward on the same level, but when they capture, they may not stay on the same level. In other words, Pawns are move in one dimension but capture in three dimensions.

  - For example, from 4e4, a White Pawn could capture at 3d5, 5d5, 3f5, or 5f5. The normal two-dimensional Pawn captures at 4d5 and 4f5 are not available.

- Knight rules are as follows:
  - The Knight is completely and exclusively three-dimensional: it jumps two squares in one direction, and one square in each of the other directions.
  - From 4e4, a Knight can go to 24 different places: 3d6, 5d6; 3f6, 5f6; 3g5, 5g5; 3g3, 5g3; 3f2, 5f2; 3d2, 5d2; 3c3, 5c3; 3c5, 5c5; 6d5, 6f5, 6f3, 6d3; 2d5, 2f5, 2f3, 2d3.

  - Notice that the Knight cannot make its normal two-dimensional move because it must always change levels.

In summary, all piece rules from FIDE chess follow the same mathematical pattern, except for 3 dimensions instead of two on top of their classic 2d moves. The pawn and knight are exempt from this where they may not use their 2d moves, only 3d moves.

1.3.1 – IDENTIFICATION OF USERS – ABOUTS THE USERS

Although that this project is predominantly being designed for Oscar, there are other potential users. These would include other chess/ chess variant players, people who enjoy strategical videogames, those who may wish to use this as mind strengthening exercise for games, maths, logical thinking. The main benefiters inevitably will be those who will use this as they enjoy chess and hence this shall be mostly tailored to them (this is that planning goes under the assumption that the user has experience with FIDE chess).

As for accessibility, the compute power of the program must be considered. Although Oscar has a high-end computer system and up to date mobile, there may be users who do not. For this reason, the program must not be largely resource heavy and should run 'smooth' on a modest device for current times.

1.3.2 – IDENTIFICATION OF USERS – 1ST INTERVIEW

Q: What is your experience with chess?

A: I play regularly and have a 2300 chess rating (intermediate average is roughly 900). I have played a few chess variants but never 3d chess.

Takeaway: My client has a complex understanding of chess and will be able to contribute complex ideas and will be able to identify problems that I may not pick up on.

Q: With regards to the UI, what would be you be expecting?

A: I would like the board to be able to be split up so that certain or individual layers can be seen separately. I would also like to be able to rotate the board and see which pieces have been captured. Additionally, a feature to see a piece's available moves would be nice. There should also certainly be a 2D option.

Takeaway: Being able to visualise your moves is very important and as such having an easy to use, understand and visualise playing field should be a priority.

Q: What would you expect from the AI?

A: There should be different levels of AI with one that will lose to an average intelligence beginner but also one that would beat an experienced player. I would like the AI to be able to play as close to as a human would as possible.

Takeaway: A varying and competitive AI is important for play to be intriguing but also the AI should behave human-like to provide more realistic play.

Q: What time restrictions on time would you like to see?

A: Just as with FIDE chess I think there should be an untimed, standard, blitz and bullet options, however, at this time I am not sure what these exact times at this moment.

Takeaway: There should be multiplied time modes, however, until the game is played the specifics are uncertain.

Q: What game modes would you like?

A: Player vs AI, player vs player. Perhaps an AI vs AI to learn scenarios, puzzle solving.

Takeaway: Having variation in what can be done inside the game is important for the user.

Q: How would you like games to be recorded/ displayed back to you?

A: A list of blunders and mistakes at the end of a game and perhaps another feature to show the best move or something to 'teach' you. Moves should be displayed on a sidebar as they happen.

Takeaway: It is important that the user gets some form of analysis during and post-game. This is most likely going to be through an AI.

Q: How would you like this game to be available I.e. on a large store front, a website?

A: Preferably as a computer app as I feel a website would be too slow but would be fine if it is not slower. Perhaps also a phone application.

Takeaway: The user seems to be happy with most forms that it could be available in, however, they seem keen to play it on a computer.

Q: What hardware/software restrictions may affect the making of this project?

A: I think that this program should be able to be ran on a mid-range system (this may affect the AI). Like a phone.

Takeaway: The AI nor UI can be so complex that an average device will struggle to compute the game.

Q: What additional features above gameplay with an AI would you be expecting?

A: Player on player gameplay - on a LAN perhaps also on a WAN although it is not essential. Less layer capabilities.

Takeaway: The user just wants a way to play against other humans too. The user would like to be able to play on a smaller board (this could create complications for the AI).

Q: What/if any two player capabilities would you like?

A: Just standard play with different time limits

Takeaway: No extras needed.

1.4.1 – PROTOTYPE (RL) – PROBLEM WITH RESEARCH AND FOLLOW UP

Due to the nature of this project with this being the first product of its kind made there are no available versions for me to try and research. Therefore, for my primary research I create a real-life prototype of the game. Other research will focus on FIDE chess and can be seen in 1.4 Research.

1.4.2 - PROTOTYPE (RL) – SHOWCASING THE PROTOTYPE

As shown above the prototype was made from 8 8x8 board lined up to represent the cube.

1.4.3 - PROTOTYPE (RL) – WHAT I FOUND AFTER PLAY

I believe this was the most important section of analysis for my project as it greatly helped both myself and the client to gain a better understanding of the game.

The main issue that I found after playing was the time with games varying from 4-6 hours long during free play. This poses a question about whether this could be boring for users. A possible solution is to reduce the number of boards used in play. My main idea was to play on four boards with one checkable king as opposed to 8 (same rules can still apply otherwise).

Thinking time and possible moves was another possible problem that arose in my mind. Due to the increased size of play it is unreasonable for a player to look at all the moves available (in fact we only look at a miniscule proportion of possible moves) – this could create problems with the AI being playing at massive skill difference with only a 2-move look ahead. Aside from the AI players will have to be competent with their ability to thing logically as the skill curve gradient for this game will be immense.

Check mating is extremely hard in this game due to the nature of the king's movement, namely: with pieces no being able to 'corner' the king so easily due the 3d nature of the board and, it is easy for the king to 'run away' through different levels. This does make the end game more intriguing for a human, however, coding an AI for this will be difficult as it is a very intuitive process. Something to consider would be to use deep Q reinforced learning or simply a neural network for the AI. Training time could be a limiting factor if I decide to take this route.

The last notable point is that board position is very different from FIDE chess. It is hard to determine where the 'powerful' positions on the boards are compared the FIDE chess where you can determine that the centre of the board is the most powerful. The again will bring problems when coding the AI – it will take a lot of playing from me to try and determine where these 'powerful' positions may be.

1.4.4 - PROTOTYPE (RL) – 2ND INTERVIEW WITH CLIENT AFTER PLAY

Q: How/would you change the overall time of the game?

A: Different time modes as mentioned before and perhaps game modes with less layers and only one checkable king.

Takeaway: Time is definitely an issue and I need to set objectives that allow for shorter gameplay.

Q: Do you think there should be a limitation on thinking time for the human?

A: Yes, of varying amounts of the user's choice.

Takeaway: There should be an ability to create different time mode modes with different starting times and added time.

Q: Do you think there should be a limitation on calculation time for the computer?

A: There should be different levels of AI that do fewer calculations.

Takeaway: Having varying AI levels is important not only for fun gameplay but also for competitive gameplay. (Quick calculating AIs and more complex AI)

Q: How strong should the AI be and what should it consider?

A: It should be strongly capable of beating the best of players but with abilities to reduce its own ability. It should prioritise piece taking as opposed to board position – the exception to the this is where the piece will be taken by a lower value piece if moved to a location. Try to defend or move high value pieces

Takeaway: A strong AI is important to the user but equally important is one that can vary in ability. After understanding the core principles of the game, both I and the user have discovered that for 3D chess taking pieces early is essential for a game to conclude past as board position matter little compared to FIDE chess till near the end game. As such, this should be factored into the AI.

Q: How/would you change the end game and/or how the game can be concluded?

A: Time runout with piece points determining the winner or classic timed play.

Takeaway: This would allow for competitive gameplay with more positioning thinking without the dread of taking the game on for several hours.

Q: Are you happy with how the pieces move?

A: Yes, the 3d capabilities make it much better than Strada.

Takeaway: The rules should not be changed.

Q: Do you have any other suggestions?

A: A help screen

Takeaway: A way to let new users know how to play is important.

1.5.1 – RESEARCH – ISSUE WITH RESEARCH

As of the date of writing (26/05/22) there are no chess variant games that play through 3 dimensions provided through mainstream online storefronts. (Specific sites checked include steam, epic games, Microsoft store, apple store, google play store).

As a result of this I will conduct most of my research through FIDE chess, commenting on any changes necessary for it to be applicable for 3D chess

1.5.2.1 – RESEARCH – EXISITNG SOLUTIONS – WEBSITES – CHESS.COM

With over 70 million users Chess.com is the most used chess app or website in the world for FIDE chess and is used by many grandmasters daily to train. Shown below is UI used for a match only as this is the only feature I am interested UI wise.

The chess pieces are shown in two dimensions which is something that I should consider alongside a 3D environment option. This can simply be implemented (from a 3d version) by unstacking the layers and changing the 3d objects for sprite images or simply flat objects. In general, I can implement both the 3d and 2d board through a 3d array of positions with piece titles (FIDE chess notation) in the 3d array. This may make it easier for users to visualise moves.

Also show are the orange arrows. These are for pre-moves where a user (in this case using a right click) can plan moves ahead visually. This will be particularly useful in my project as planning is used to an

even greater extent than in FIDE chess. I can implement this by arrows that are either object that can be enlarged and translated to show the respective move the user has ‘pre-moved’.

Additionally, seen on the right, is a move log in which previous moves are shown to the user. This is something that my client stated they would be interested in. As an extension of this feature there is also a ‘move rating’ for each move made where it shows how each moved affected the position of both black and white. This may be something worth implementing in my game to aid learning, thought this will come at some computational expense. Implementing the past move list can be done simply by taking the name of the piece moved from the 3d array of positions and the starting and end position in said array, then displaying this to the user via text (and in this case a sprite for the respective piece).

One small detail to look at and compare to other FIDE chess games is the colour of the tiles. Whilst it may not seem important, when it comes to 3D chess being able to easily distinguish what tiles line up makes planning easier and hence planning time shorter. The green and cream colour scheme works well as both contrast each other and the black background. They also include an option to change the ‘theme’ of the board. This is not of the upmost importance, but it would be easy to implement by simple colour changes of the board and piece objects.

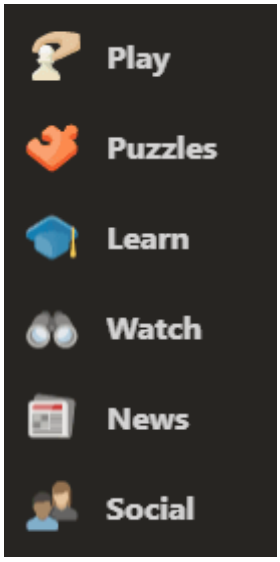
Aside from the UI there are other features that I investigated. Chess.com allows two ways to play using a chess engine, these are the analysis board and ‘playing against AI personalities’. The analysis board uses an AI engine, Stockfish -1.5.3.1, where you can play against yourself or another person on the same device while the AI analyses and feedbacks on the moves made. This can be seen above.

As for playing against personalities, Komodo – 1.5.3.2, is used where personalities can be replicas of how celebrities play, certain play styles i.e., aggressive, passive, or adaptive (to how you play) as well as being able to vary ability.

Both above would be something I am interested in looking into, practically it may be infeasible to implement the former exactly as the number of moves available in 3D is exponentially larger than in FIDE chess. This would mean to rank all possible moves and how they affect the future game would take too much computational power. I could still use this feature with and algorithm that utilises a heuristic approach with approximate worst, best and mean values or by rough standard deviations (make assumptions for N, mu and xi).

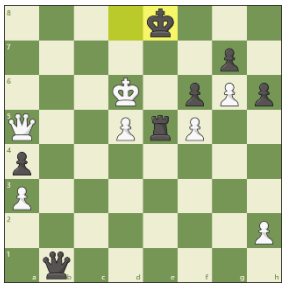
$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

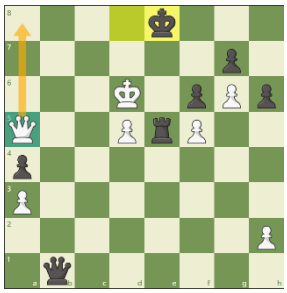
σ : Population standard deviation
x : Datapoint value
μ : Population mean
N : Population size

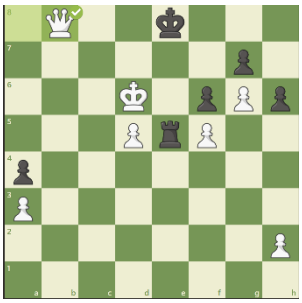


Shown to the side are the other options that Chess.com. Most of these functions will not be useful for my project. Social may be of use if the game is to be played across a WAN. News will not be relevant as the news function here depicts new chess information. Watch allows a user to see AI vs AI or Grandmaster games, which is not necessary.

Puzzles are the other main feature and is also one my client said they would like to be included. Puzzles work by setting out scenarios that may occur in a game and you must reach the goal (usually checkmate) in the least moves possible. An example is showcased below (3rd picture shows what using the hint button will do. These are useful for brain training and learning specific situations better. To implement this, I would have to create my own scenarios by storing sets of 3D piece position arrays alongside the set of moves that achieves that best solves the puzzle and loading them upon user request.

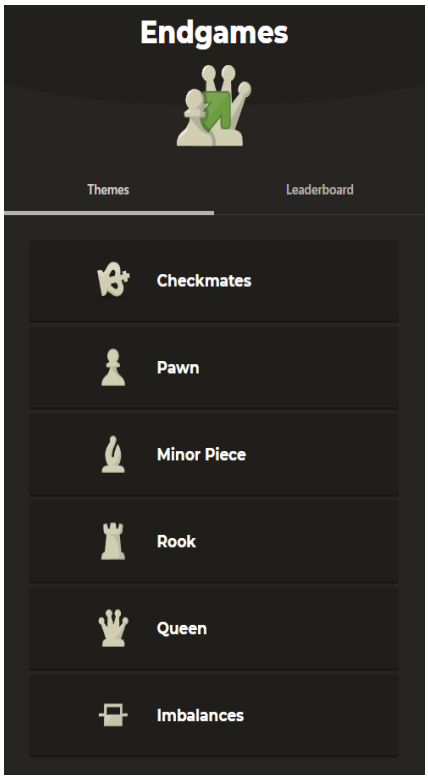


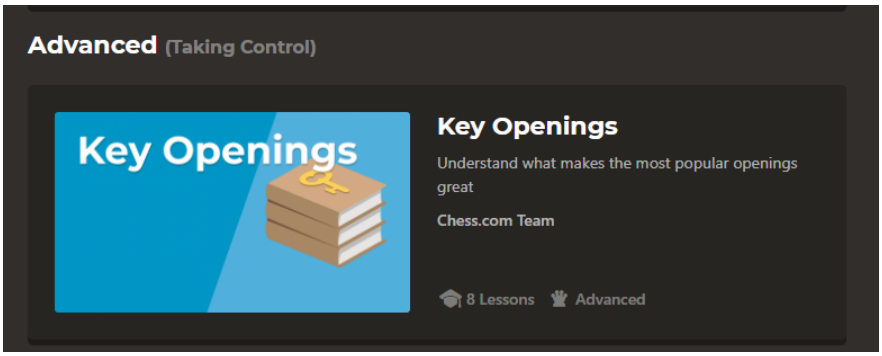


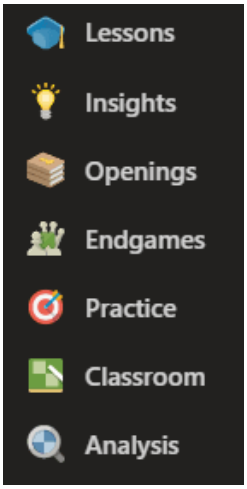


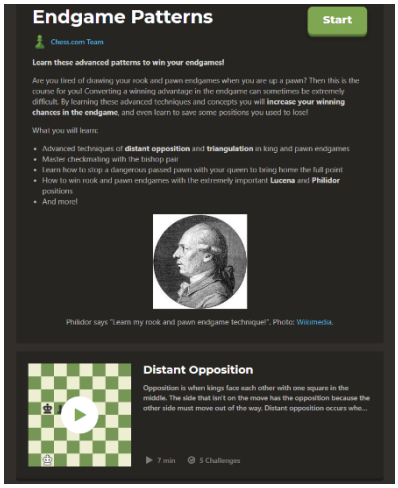
The other section of interest for me is the learn function where words, pictures and videos are used to explain how to play the game for different levels i.e. basic, intermediate and advanced strategies and plays as well as instructions on how to play. There are also additional functions such as the 'classroom' where you can be taught be a real coach online, analysis of any game played

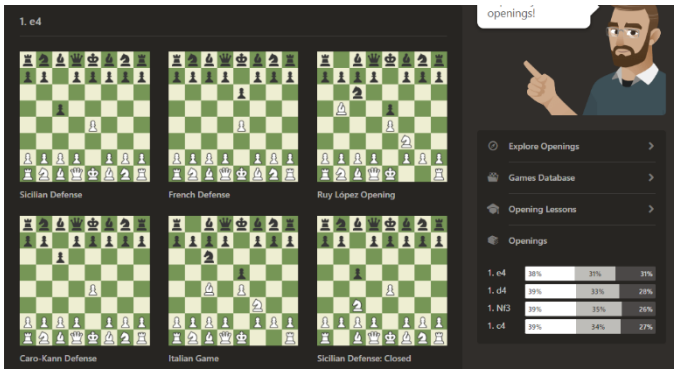
by any two players, a database of opening, insights into any players chess history and endgame techniques. This can all be seen below.

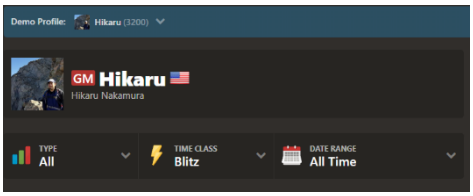


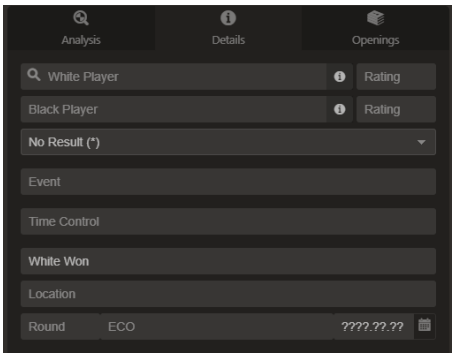










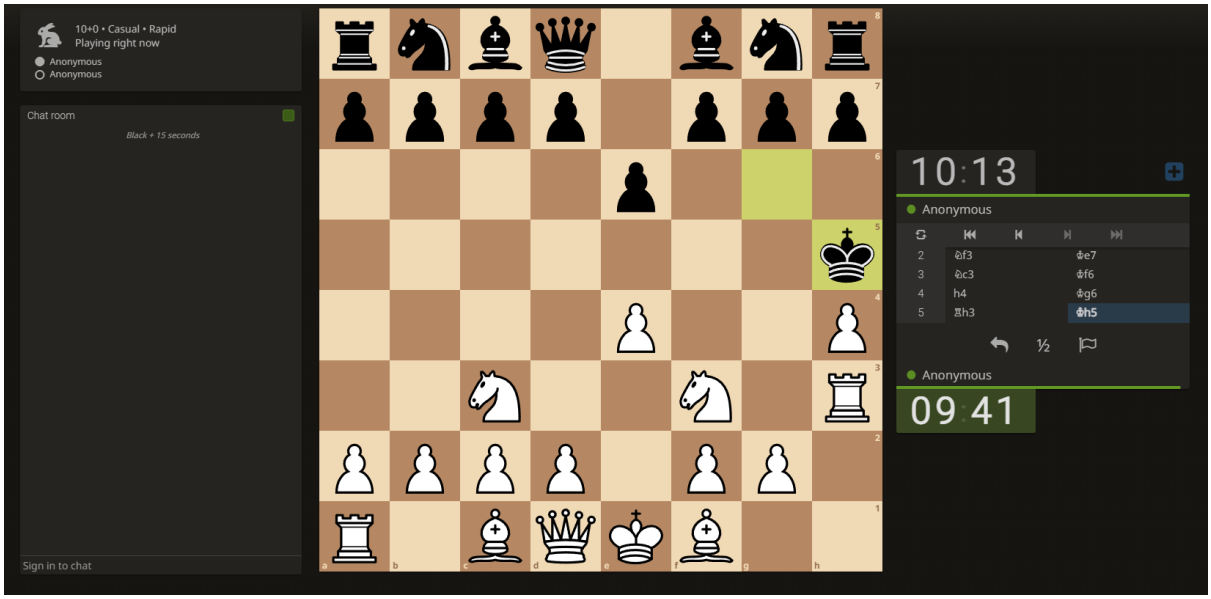


Lastly, Chess.com supports all FIDE chess time modes, (found at the link below), however no option for custom time rules. This is something I would want to use in my game as time plays a very large factor with how intriguing the gameplay is. https://www.fide.com/FIDE/handbook/LawsOfChess.pdf

1.5.2.2 - RESEARCH – EXISITNG SOLUTIONS – WEBSITES – LICHESS.ORG

Lichess.org is a site the Oscar uses frequently. It has roughly 15 million users with grandmasters using it too, making it one of the most popular chess sites. It is worthy to note that through my research for this I will not be going to depth on any features that are the same as that in previous research.

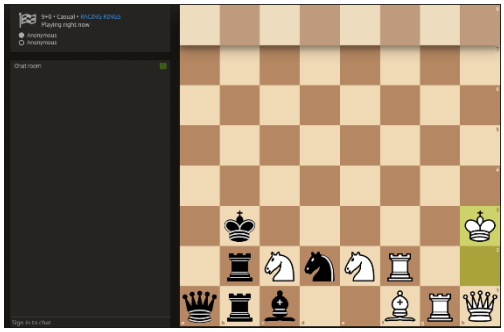
Shown below the UI when playing a game against another player.



The main and obvious difference here when compared to Chess.com is the chat function on the left-hand side. This, if play across a WAN is to be used (decided upon later), would be a useful function. There are some issues with live chats however regarding abuse. I am not, thought, too concerned with this are chess players are not typically people to abuse the opponent due the rules and etiquette associated with FIDE chess. For this reason, if I were to implement this function there would only have to be very simple forms of chat moderation. The other two noteworthy differences are the board colour and lack of sprites in the past move section. Although the 'classic' chess colours are used here I would still opt for the colours used in Chess.com as the default as I found it easier to plan moves with the higher contrast. I will also create other themes that the user can select with classic being one of them. As for the lack of sprites, the formal chess notation is necessary, however, I do feel the sprites quickly and easily identify the past moves. For my project I will implement both alongside each other with an option to hide the sprites.

For AI, lichess uses its own neural network chess engine called 'maia' which works off three levels (1,5,9). It was trained over 10 million games between over 1000 players. This is unusual to see from a chess site as typically chess sites used external chess engines. This could be something that I can implement with respect to my engine, though there may be an issue with an appropriately large set of training games.

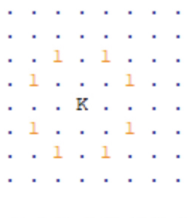
In addition to standard play lichess includes basic chess variants, such as: antichess (if you can take a piece, you must), chess960 (where pieces starting place is randomised) and king race (each player must try to get their own king to the 8th rank as fast as possible). These can both be seen below. Although my game is a chess variant it is impractical to implement this as my play space is a cube, unlike any other variant. Aside from this, Lichess offers the same puzzle and learn options as Chess.com.

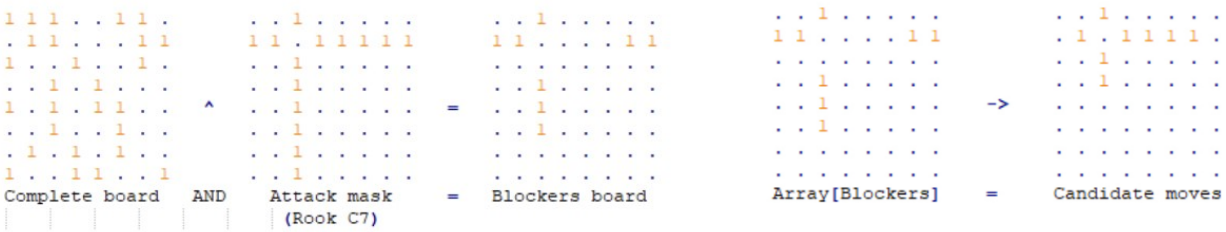


Regarding the puzzles, however, lichess I believe has a better deign. Its puzzles can be derived from previously played games and tailored to how you play specifically (if you make a free account). Additionally, it shows the moves taken to reach the current board position which could also be helpful. These are both things I would wish to implement in my project. I can do the former by sorting all previously played games in a database and choosing stages form those games and do the later through taking the moves from those games and displaying them.

1.5.3.1 – RESEARCH – EXISITING SOLUTIONS (AI) – STOCKFISH

(Note ranks(A-H) are rows and files are columns (1-8)). Stockfish utilises bitboards to store the state that the board is in. The 64-bit variable can store the position of specific pieces in a game (as a board is 8x8 squares that can be occupied). A one at a specific position shows that the piece that the bitboard is representing is at that location. Bitboards start from the lowest left square and work across the board then up the ranks. (A bitboard for the starting position of pawns is shown to the side). Therefore, a move of 1 bit left would be left shift 1 and a move up one square would be a shift of 8 bits left. It is worthy to note bitboards here are stored using little-endian coding and hence empty squares are only not shown if they come after the lowest, leftist piece.

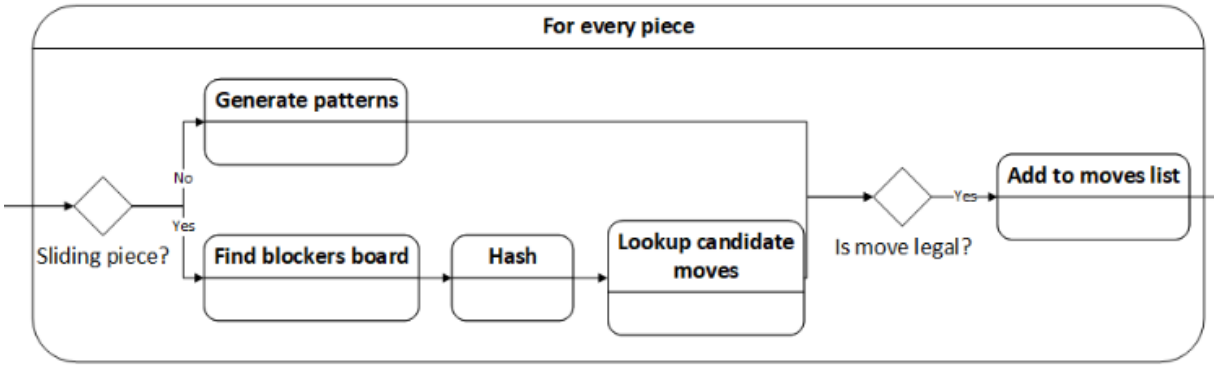




Constants are also set for pieces that have specific limited movement patterns; these are called candidate moves and are stored as bit shift operations. (Shown for a knight are shown above to the side). This representation then allows the AI to find all possible moves and starts to make a tree of pseudo-legal moves. These moves are then checked under various verifications. Firstly, the edge ranks and files are masked that the candidates that go outside the bounds of the board are determined illegal, then blocking pieces and discovered checks (king is under attack) eliminate other illegal candidates. The pseudo-legal moves can now be determined legal or illegal and the tree (for the moves with only pattern candidates) will be correct.

As for pieces that may 'slide' across the board, it is more complicated. The 'rays' that they may slide across are plotted onto a bitboard then an AND is applied to the attack rays board (Attack mask) and the complete board position (a combination of all piece bitboards) to produce a board where pieces interrupt the rays. To save computational power Stockfish indexes specific bitboard combinations and relates them to premade candidate moves. An array cannot be used, however, as this would be too large to store in memory with the board being 64 tiles (this means the candidate array will have 2^64 elements). For this reason, a hash map is used, and the more manageable issue becomes finding an efficient hash function.

For this, magic boards are used where the bitboard of blockers is multiplied by a 'magic number' and then shifted by the number of bits gained to produce the hash key. This magic number is calculated by a function given the piece and its current space. All magic numbers are calculated when stockfish is first run, and it is actually done by brute force as opposed to a mathematical function. Shown below is an overview of the process discussed.



The implementation of this method of determining the board state may be harder in 3D due the increased since and hence increased calculations involved. If I wanted to do it this way, however, I would have to change the bitboards to 512 bits as opposed to 64. Then, create more patten bitboards and have a larger store of bitboards for sliding pieces. Additionally, I would need extra masking for the bitboards for the pattern moving pieces.

Once a tree of moves is created Stockfish must determine which move it would like to make, this is where the different levels come in to play. For Stockfish 12 and above it uses a NNUE (more on neural networks in 1.5.4) to determine the move. The inputs are Boolean as follow: (where C can be friendly or enemy) is piece A on square B with a C piece D on square E. This is iterated over for every possible piece combination on the board making up to over 40k inputs for each piece (each A – meaning without iteration of A). Some of these can be eliminated, however, using a function that detects redundancy leaving only moves that that correspond to certain criteria per piece. Stockfish also uses binary encoding

reducing this down to under 800 and has the added benefit of incremental calculation through hidden layers. The advantage of an NNUE for me specifically is that it is optimised for CPU computation without the need for a GPU, allowing the AI to run on devices without a GPU. The features that benefit everyone, including myself, are the ability to use incremental calculation and the encoding for inputs.

The specific setup for Stockfish 12's neural network has three hidden layers and specific, heavy optimisation. The neural network is trained from a large data set of known moves created from Stockfish classic (below 12) where randomly generated positions are linked with the optimum subsequent moves by Stockfish classic. This is done with a very low-depth (low move lookahead – talked about later in this section) to create a large database in a reasonable time frame. Once trained to a reasonable standard, the neural network can be further trained used more complex, specific examples.

For Stockfish classic, an evaluation function is created from different chess concepts that eventually become weighted. The main concepts are material – consisting of material imbalance: total number of pieces for players, material advantage: the power of pieces in play and the power of combinations of pieces in play. The king – king safety: upcoming checks, current checks, protecting pieces, surrounding pieces. Space – positional advantage: having pieces attacking or occupying certain positions, occupying space: which squares are occupied by pieces. Threat – pieces under threat and protected, hanging pieces (under threat with no current protection), king threats, pawns or pawn line pushing, discovered threats, moves that will result in Exchange or threat, pieces not under threat but in a weak position. Strategy – pawn advantage: attacked pawns, pawns close to promotion, coupled/doubled/isolated/protected pawns, other piece advantage: blocked pieces, pieces in good positions for their move time, batteries of rooks and queens (same rank or file), enemy king under attack, rook on an open file, bishop on long open diagonals, bishop on long diagonals, bishop x-ray attacks.

There are two different ways the above must be weighted. Once for midgame and one for endgame as they are very different types of play. (There is no early play as this would simply be an opening). The weighting for each one is still evolving with chess programmers making pull requests to improve Stockfish's ELO rating. These two sections, however, are do not have a distinct start and stop point. Therefore, Stockfish interpolates these two weightings from 0-128 and with respect to the 50-move rule, that if in 50 moves no pawn has moved or piece been captured the game results in a draw. Once all this information has been weighted it is run for two looks ahead to discover any extra bonuses or penalties. Finally, the evaluation for the best move uses alpha-beta search (decreases the nodes needed to be looked at for minimax rule) which will determine the best move according to minimax rule using static evaluation (this is that to assign and sum values for aspects of the game - listed in the last paragraph). (Most chess engines use alpha-beta pruning.)

Despite Stockfish's (12) extreme optimisation of its neural network's inputs, the issues still arise in 3D chess of computational power and whether using this method would be viable. If this is an issue, I can adopt a similar but less complex scheme to Stockfish classic: creating a function from weighted parameters. In doing this I can chose more precisely how 'good' the engine is and how resource intensive it is.

1.5.3.2 – RESEARCH – EXISITING SOLUTIONS (AI) – KOMODO

Komodo and Stockfish are typically named the top two chess engines in the current day and are still 'fighting it out'. Newer versions of Komodo are commercial (unlike stockfish), however, older versions

are publicly available. Komodo uses Stockfish's library to parse moves and positions and so will always be limited by Stockfish, however, this does mean it does have an advantage in that is doesn't have to compute this. Komodo also has adjustable parameters for percentage of time spent on analysis and how many evaluations it makes before analysis stops.

Komodo works of programmatic chess where it will determine the best moves and most advantageous positions for both side and then use probability to choose the move to play. This is done by looking at techniques used and common chess knowledge it has learnt from previous matches. This and the fact that it attempts to use a small amount of memory creates an aggressive play style as aggressive players value simpler play, trying to shut down complex positions and in this forcing out mistakes from the opposition.

Komodo must be 'trained' in a specific way where it must draw and lose to weaker opponents to update its evaluation function. This is usually done through playing against other opponents. The reason that Komodo is good is that it is adaptive to new situations that arise on the board which is something than humans can do but other chess engines struggle to do.

This approach, in my opinion is not suited for this project. Komodo's adaptive abilities make it significantly harder for a human to beat. If I were to implement this, it would mean creating a greater handicap for the engine against a player and the extra head room gained would not be needed as human 3d chess players would not surpass the limit without the addition of adaptive play.

1.5.3.3 – RESEARCH – EXISITING SOLUTIONS (AI) – KOMODO'S DRAGON

Dragon is a unique engine as it combines both programmatic techniques with the use of AI (in the form of an NNUE). In doing this dragon can analyse to a high level whilst also maintaining the adaptability that comes with being able to determine the probability of winning after a move.

Unfortunately, this is not publicly available and therefore cannot investigate the inner workings. It is worth noting that this type of complexity is unlikely to be needed as due to the size of the board in 3d chess computers will automatically be better elo for elo than in FIDE chess. I.e in a match of 3d chess between a 1500 elo chess player vs the compute power of a 1500 elo chess AI the AI has a much better chance. For this reason, the AI used for 3D chess requires less advanced chess techniques to beat a player.

1.5.3.3 – RESEARCH – EXISITING SOLUTIONS (AI) – DEEP BLUE

Deep blue was an extremely large parallel system designed to be able to create a graph/tree of all possible moves that can be made. Deep blue is a system with many components. I will be looking at deep blue for ideas to implement rather than a way to convert it for 3D chess.

After reading of the workings of deep blue, all that I saw that I would want to implement its idea of fast and slow evaluation, where in times where a simple approximation is only required the score of evaluation is based on a piece's specific position. In slow evaluation, other key chess concepts can be involved. Not only is this an easy way to save compute power, but it also brought up another idea for me. That being different levels of AI can be created by simply removing more complex chess concepts from my evaluation algorithm. This is a particularly good idea as different levels of AI will play like

different levels of humans (as humans better at the game tend to know and understand more advanced chess concepts).

1.5.3.3.1 – RESEARCH – EXISITING SOLUTIONS (AI) – ALPHAZERO – DEEP MIND

Created by Deep mind, later brought by Google, AlphaZero is a generalised engine, not just for chess where it will take the rules of a game, train against itself and become better. It is an advancement on AlphaGo Zero, an AI to play Go given 'zero' extra information beyond the rules of the game. This makes AlphaZero vastly different to the likes of stockfish or Komodo that are built of human experience and more of an 'evolutionary strategy' where the algorithm and AI are adapted by humans to become better.

AlphaZero consists of a neural network and the MCTS (Monte Carlo Tree Search). It is worthy to note that these are now used in many other chess engines. The neural network will receive the layout of the last few boards and outputs, to the MCTS algorithm, the probability of winning with some 'best moves. The MCTS will then spend greater time 'researching' not only the best moves but also random moves in an exploration-exploitation strategy.

The genius behind AlphaZero is how it can be trained, and this is due to the MCTS still looking down paths that are given by the infant neural network. The results, percentages given by the neural network and the searching by the MCTS can then be made into new weights, which creates a better NN, creating a better MTCS. This creates a rapid feedback loop allowing the network to train extremely fast (as far as NN training goes).

1.5.3.3.1 – RESEARCH – EXISITING SOLUTIONS (AI) – ALPHAZERO – LC0

Leela Chess 0 or LC0 is a publicly available version of AlphaZero. It was written by an amateur programmer and trained over many system (and still currently being trained) by many people online who wish to. Although AlphaZero does not have an elo rating, LC0 is thought to be more powerful.

Unfortunately, it is unlikely I will ever manage to fully train a system like this to the level that LC0 and AlphaZero are capable of without the same public training scheme used for LC0. My AI would not need to be anywhere near as powerful, however, and so this could be something that I can implement in my project simply by changing the inputs nodes for ones capable of understanding a 3D board and use less previous games.

What makes this especially desirable over the previous engines is its ability to self-train. The desirability comes from the fact that I have no 'model' games that can be easily be given the other neural networks to train them. (More on NN in 1.5.4.1.)

1.5.4.1 - RESEARCH – MACHINE LEARNING – Q-LEARNING AND MDP

Q-Learning (modelled of MDP – Markov decision process) is a simple type of reinforcement learning and so I research for this project. It is one in which an agent takes actions in an environment for each time step and with that the environment may update to a new state and a reward may be given. As part of agents there are policies (pi) which map certain states (of the environment) to probabilities of choosing a certain action from that state. Alongside this are the value functions: V value function– gives every state a value under pi, Q value function– gives every action a value under pi. These create q values which give the value of a state-action pair (making a particular action and a particular state). These values are then optimised (where each state action pair yield the highest reward possible) using a

reinforcement algorithm and when the optimal values are found for each state action pair (q*) then the optimal policy for the agent to follow is also found. This reinforcement algorithm will consist usually by not always solely the following:

- Episodes - sections in which learning occurs and between episodes the environment is reset to a random state or set state
- Bellman Equation for optimality – how q values should change
- Cumulative reward and expected return – total reward for an episode and how much is expected under a policy
- Discount rate and discounted reward – gamma (how much less future rewards are worth during calculation)
- Epsilon greedy strategy - an exploitation-exploration trade off
- Learning rate – how much old data is retained for q values

If I were to adopt this method of learning and simply try to use every single combination of how pieces can be arranged on a board as the states there would be an enormous amount, that combined with every possible action for those states creates and unfathomable number. Additionally, the reward system associated with MDP is not suited to this type of game as the reward would come after >100 moves.

1.5.4.1 - RESEARCH – MACHINE LEARNING – NEURAL NETWORKS

'Neural network' is a phrase that appeared numerous times in my research and appeared to be the key for how many of the chess engines worked. For this reason, I took a deeper look into them.

Neural networks work by connecting nodes to each other with weights and biases, where weights determine how much one neuron affects the one after it and biases are a threshold amount for a neuron to activate. This weight and bias layout allows neurons to be described as functions but also and more importantly allows them to compute anything. (This can be proven as specific weights and biases allow a NAND gate to be created – a universal gate.). In most cases reLU, leaky reLU or the sigmoid squishification function are used modify the state of the neuron to help with learning The layout for a neural network consists of an input layer, hidden layers and an output layer and the activations are calculated by matrix multiplications.

What allows the neural networks to be so powerful is their ability to learn. This is done though minimising the cost function that is calculated by the mean square loss between the wanted result and the one obtained from the neural network. By finding the gradient of the cost function (through differentiation using the chain rule) local minimum of the cost function can be found and at this point the obtained and wanted result should be extremely close. One important other addition that may be of use to the project is the use of mini batches for stochastic gradient descent, where smaller batches are trained on creating slightly more random but faster gradient descent (faster learning).

For this project if I decide to use a neural network, I will most likely adopt a similar approach to one of the Chess AIs for FIDE chess that use a neural network (where the neural network is combined with other moving parts or a very specific type of NN is used I.e NNUE).

Chess Engines that use NN are at 1.5.3.X || Discussion on if and how I will use a NN at 1.6.X

1.6.1.1 – RESEARCH – PRACTICALITY FOR AI – REINFORCEMENT LEARNING