

Creating an AI to play Three-Dimensional Chess

By Alex Mann

This project develops chess into the third dimension, first as a 2-player game and then adding an AI opponent.

2D chess rules:

The rules of 2D chess on an 8x8 board are well known, with the objective being to take the opponent's king, and with permitted moves as summarised below:



Fig 1.1 – board layout



Fig 1.2 – pawn moves



Fig 1.3 – rook moves

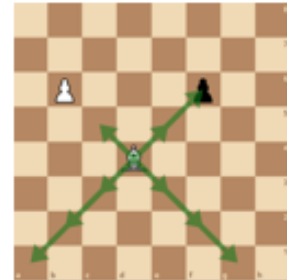


Fig 1.2 – bishop moves



Fig 1.5 - knight moves



Fig 1.6 - queen moves



Fig 1.7 - king moves

Phase 1 - Researching and evaluating rules:

As 3D chess has no official rule set or any documented matches, the rules must be decided. Many people throughout history have theorised and suggested different formats, therefore I researched various existing approaches in order to discover the possibilities for rules. My aim for the rules was to make the game as intuitive as possible as an extension of regular chess, so rules should be similar wherever possible, maintaining the advantage of attacking play and avoiding too many draws. Table 1 summarises each game researched, their specific rules and my evaluation.

Table 1:

Version – Creator, Year	Notable rules	Evaluation
Kubikschack – Kieseritzky, 1851 ¹	<ul style="list-style-type: none"> • First recorded mention of 3D chess • No documented rules, but used 8x8x8 board 	-
Johnson’s Three-Dimensional Chess – Rick Johnson, 1966 ² Chess in the Third Dimension – Skor-mor, 1976 ³ Strato Chess – Dynamic Games, 1973	<ul style="list-style-type: none"> • 8x8x3 board • Pieces move normally, plus 1 square up or down 	<ul style="list-style-type: none"> • Simple and intuitive • Not truly 3-Dimensional as no variation in movement between layers. • Limited strategic possibilities therefore potentially less-interesting gameplay
Hagemann’s Three-Dimensional Chess – Wally Hagemann, unknown ⁴	<ul style="list-style-type: none"> • Knights move by vector (2, 1, 1) • Rooks move in 1 direction • All other pieces move in a similar way to Johnson’s 	<ul style="list-style-type: none"> • (2, 1, 1) vector knight movement seems an unnatural extension of (2, 1) 2D move • Rook move is both loyal to its regular function and truly 3-dimensional
Raumschach – Ferdinand Maack, 1907 ⁵	<ul style="list-style-type: none"> • Most widely played version • Began with 8x8x8 board • Settled on a 5x5x5 board with a unique setup over 2 layers per player, including new piece called a unicorn • Pawns move forward either horizontally or vertically • Rooks, bishops and unicorns move in exactly 1, 2 or 3 directions respectively • Knights move by a (2, 1, 0) vector • Kings and queens move in 1, 2 or 3 directions at once • Pawns cannot move 2 on the first move as the board is too small 	<ul style="list-style-type: none"> • Choice of bishop, rook and knight move seems to best embrace the three-dimensional aspect while remaining intuitive • Optional unicorn is interesting as a unique 3D addition • I think the queen should be limited to two directions to avoid having too much power • King should also be limited to two directions to allow queen-king checkmates • 5x5x5 board gives satisfying cubic shape with reasonable size, but unusual setup makes it too different to regular chess

While an 8x8x8 board initially seems ideal, it leads to a strange and unnatural opening. Pritchard concluded it is “the most mentally indigestible for the players ... Less demanding on spatial vision, and hence more practical, are those games confined to three 8x8 boards and games with boards smaller than 8x8”⁶, I therefore decided to adopt an 8x8x3 board with the layout in Fig 2.1 for simplicity, settling on moves illustrated below:

¹ Anthony Dickins, *A Guide to Fairy Chess* (New York: Dover Publications Inc., 1971), 16-17.

² David Pritchard, *The Classified Encyclopedia of Chess Variants* (n.p.: John Beasley, 2007), 225-233.

³ Ibid.

⁴ Ibid.

⁵ Dickins, *A Guide to Fairy Chess*, 16-17.

⁶ Pritchard, *The Classified Encyclopedia of Chess Variants*, 305.

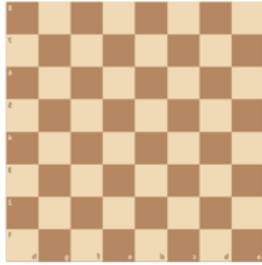


Fig 2.1 – board layout

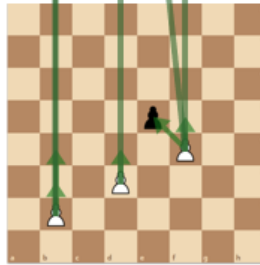
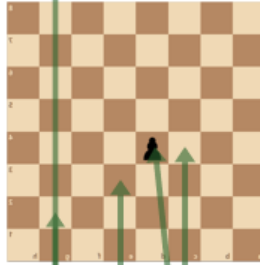
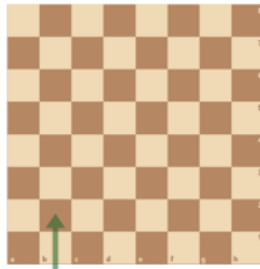


Fig 2.2 – pawn moves

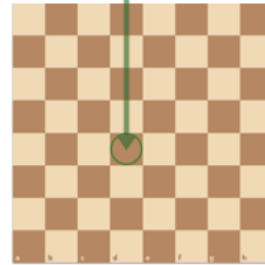
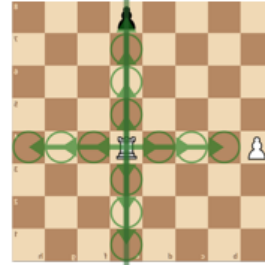
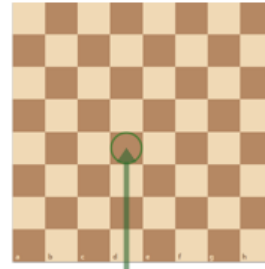


Fig 2.3 – rook moves

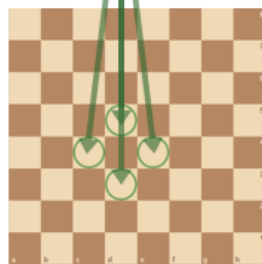
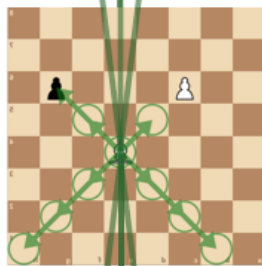
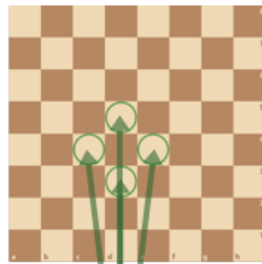


Fig 2.4 – bishop moves

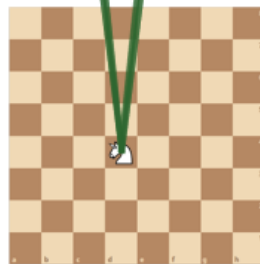
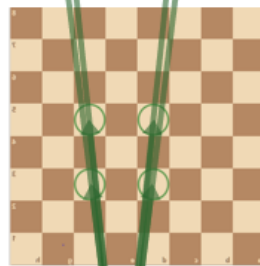
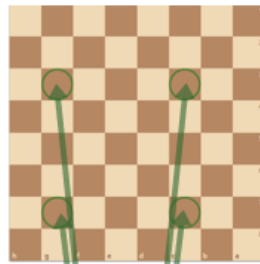


Fig 2.5 – unicorn moves

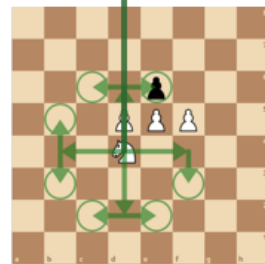
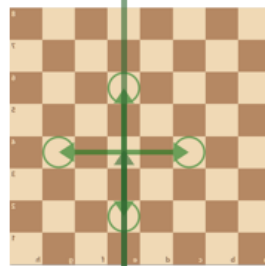
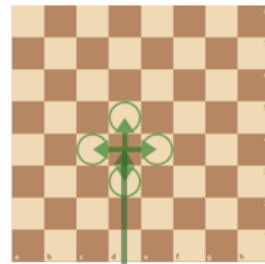
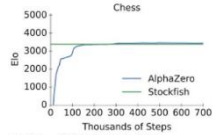


Fig 2.6 – knight moves

Phase 2 - Researching AI options:

With rules agreed, I examined and evaluated methods of creating an AI chess player as summarised below:

AI method	Summary	Evaluation
Deep learning from match database	The usual approach to all modern chess AIs, data from thousands of matches is analyzed to teach a program which moves are beneficial	Since 3D chess is not widely played, there are no games to analyze so this approach is not possible
Deep learning from first principles ⁷	A program is given the rules of the game and plays millions of matches against itself to learn the best ways to win	<p>"5000 first-generation ... and 16 second-generation TPUs were used to train the neural networks ... for approximately 9 hours."⁸</p> <p>With the 179600GB/s bandwidth used here⁹ compared to my available 40GB/s and using the data from Fig 3.1¹⁰, it would take me 1150 hours of continuous computing power to reach the level of a casual player, so would be impractical</p>  <p>Fig 3.1 – AI skill level over time</p>
Monte Carlo Tree Search ¹¹	The algorithm uses random explorations from each possible move to decide on the best one	Uses a method called backpropagation to store the various game positions, so is only efficient when the same positions are reached frequently, which is not the case with chess
Monte Carlo Minimax Search	The algorithm searches a set number of moves ahead, evaluating each possible outcome with an evaluation function, and makes the move with the best evaluation. Can be optimised with Alpha-Beta Pruning ¹²	"Designed for ... games where one would rarely expect to sample the same successor state multiple times" ¹³ – this is ideal as positions are rarely repeated in chess. Preferred approach as provides a large skill gain with little computing power, especially with Alpha-Beta pruning

⁷ David Silver et al., "A general reinforcement learning algorithm that masters chess, shogi and Go through self-play", *Science* 362, no. 6419 (2018): 1140-1144, <https://science.sciencemag.org/content/362/6419/1140>

⁸ Silver, "A general reinforcement learning algorithm", 1142.

⁹ Patrick Kennedy, "Case Study on the Google TPU and GDDR5 from Hot Chips 29", Serve the Home, published 22 August 2017, <https://www.servethehome.com/case-study-google-tpu-gddr5-hot-chips-29/>

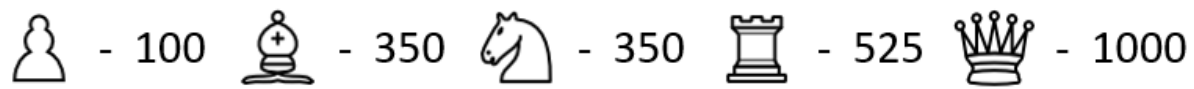
¹⁰ Silver, "A general reinforcement learning algorithm", 1140.

¹¹ Guillaume Chaslot et al., "Monte-Carlo Tree Search: A New Framework for Game AI" (paper presented at the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, California, 22-24 October 2008), <https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-036.pdf>

¹² Marc Lanctot et al., "Monte Carlo *- Minimax Search" (paper presented at the 23rd International Joint Conference on Artificial Intelligence, Beijing International Convention Center, Beijing, 6-9 August 2013), <https://arxiv.org/pdf/1304.6057.pdf>

¹³ Ibid.

I implemented the Minimax Search in python¹⁴, adapting it for 3D chess. The Minimax Search requires an 'evaluation function' that gives a point score to any arrangement of the pieces on the board. Based on successful strategies in 2D chess, I chose a function that calculates a total score per player by assigning the following scores to each piece (Fig 3.2).



Phase 3 - Testing and evaluating the AI:

In this phase, I tested the AI to analyze its performance and skill level. Firstly, I let the AI play against itself at varying search depths (looking a different number of moves ahead) to compare its abilities. As the search depth is clearly vital to the skill of the algorithm (Fig 4.1-4.6), I needed to run the program as deep as possible. I also tried a modification to the AI which I called 'curious minimax', which looks further ahead whenever there is a positive outcome to find out if the benefit is preserved, but from the investigation in Fig 4.7, it is clear this is not beneficial.

Next, I tested the time taken to decide a move at different depths, both with and without Alpha-Beta Pruning (Fig 4.1). From this graph you can see that Alpha-Beta Pruning makes a huge difference, reducing the time from 72 to 5 seconds at 3 depth.

For quality game-play, I decided on a target of 1 second per move, limiting the highest search depth possible to 2 (Fig 4.8 shows the optimized algorithm takes on average 0.1 seconds for 2 depth, and 5 seconds for 3 depth).

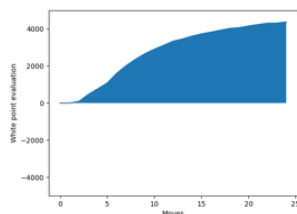


Fig 4.1 - 1 vs 0 depth - evaluation

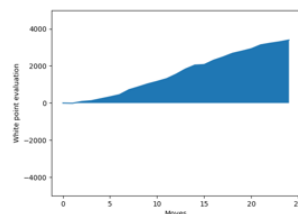


Fig 4.3 - 2 vs 1 depth - evaluation

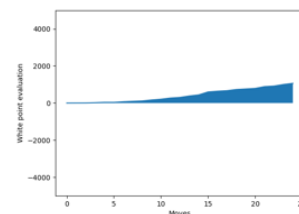


Fig 4.5 - 3 vs 2 depth - evaluation

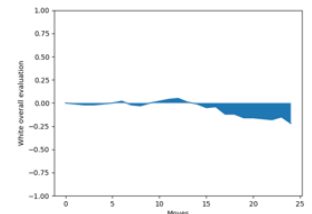


Fig 4.7 - curious vs non-curious - net win %

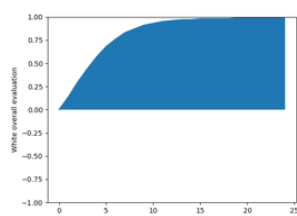


Fig 4.2 - 1 vs 0 depth - net win %

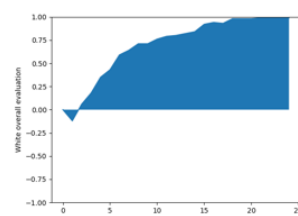


Fig 4.4 - 2 vs 1 depth - net win %

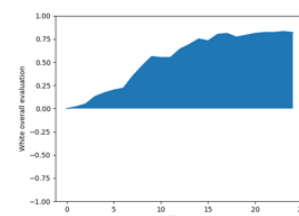


Fig 4.6 - 3 vs 2 depth - net win %

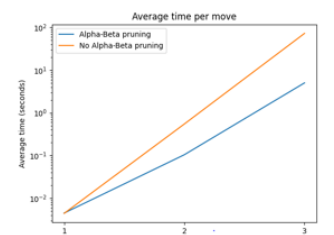


Fig 4.8 - average time per move - logarithmic

Finally, I played 10 games against the AI at 2 depth. In the first few games, I found it difficult to visualize the moves in 3D and so the AI capitalized on my mistakes and won comfortably. As I got used to the movements, games became more balanced and strategic, but while the AI was still ahead, it struggled to finish the game. In the final games I was ahead but also found it hard to win on a very open board.

¹⁴ Lauri Hartikka, "A step-by-step guide to building a simple chess AI", Free Code Camp, published 30 March 2017, <https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977>

Based on these games I concluded that the best optimization strategy would be increasing the search depth and positional awareness to avoid random moves when there were no available piece takes.

Phase 4 - Optimisation:

First, I improved the positional awareness by creating 'piece-square tables' ('PSTs'), a table of the relative extra value of each piece being in different positions on the board. I adapted these from 2D chess considering the mobility of the pieces and how this changes in the third dimension. I also increased algorithm speed by creating a new algorithm, only re-evaluating the board for the the moving piece.

Secondly, I performed the following optimization tests:

Test	Conclusion
Old v new algorithm, with and without Alpha-Beta Pruning, search depths 1-4 (Fig 5.1-5.2)	<ul style="list-style-type: none"> New algorithm has significant effect on timings at all levels Time for 3-depth algorithm reduced below 1 second target, allowing use in gameplay
10,000 games PST v no PST (Fig 5.2)	<ul style="list-style-type: none"> Advantageous after 10th move

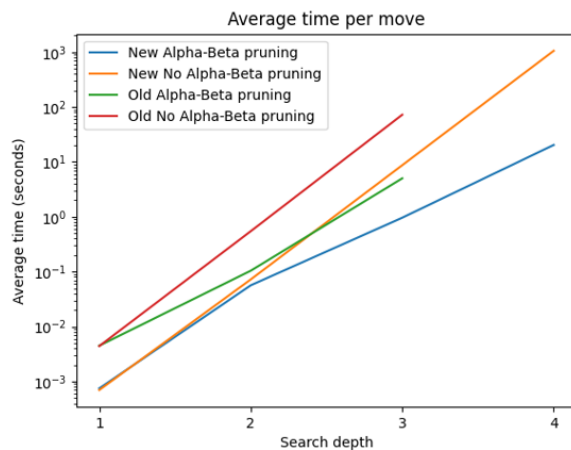


Fig 5.1 – average time per move – logarithmic scale

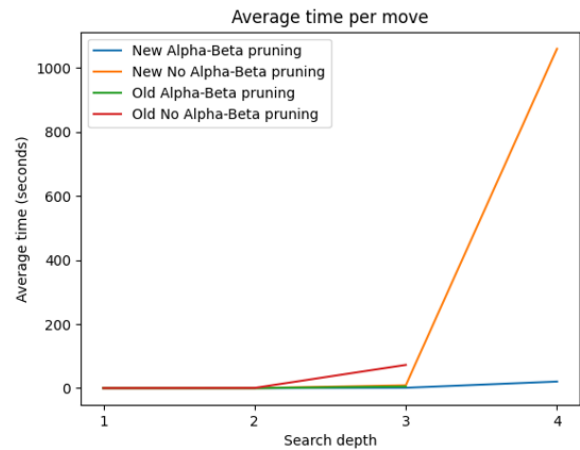


Fig 5.2 – average time per move – linear scale

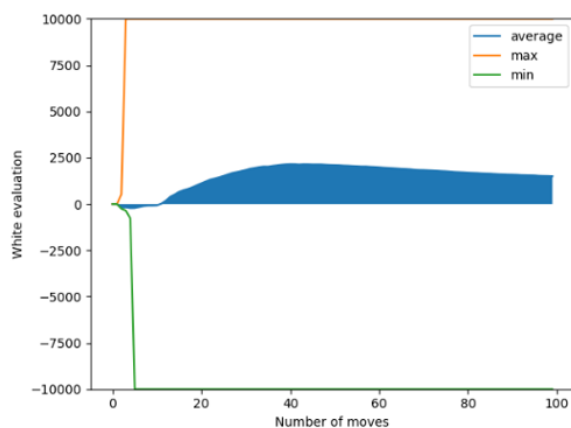


Fig 5.3 – with vs without tables - evaluation

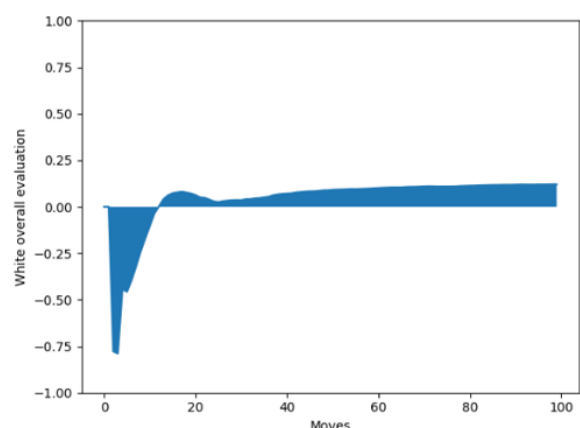


Fig 5.4 – with vs without tables – net win %

Phase 5 – Player feedback:

Before seeking player feedback, I improved user experience with extra settings and features, giving the option to experiment with different rules. This was packaged into a final product that could be downloaded and run on any Windows PC.

Finally, the AI was played against some highly ranked chess players, losing narrowly to a 1900 rated player and beating a 1700 player. I therefore estimate the the AI's skill level at 1800ELO, ranking it as a Class A player, one rank below Candidate Master.

Conclusion and evaluation:

Overall, I achieved my goal of creating an interesting and intuitive format for 3D chess together with an AI algorithm adapting multiple computing methods to produce an intelligent program. With more time and computing resources, I would have liked to use machine learning to perfect the piece values and PSTs through extensive gameplay.

I learnt a lot about AI algorithms throughout this project and enjoyed developing my analytical and programming skills as I applied them to a new and challenging game.

Finished artefact:

- Windows App – <https://www.dropbox.com/s/6y7pfm5m85a35qo/3D%20chess.exe?dl=0>
- Video of AI beating me - <https://www.loom.com/share/1f806b1a4c5047548920ed1994966a0f>

Bibliography:

1. Chaslot, Guillaume, Sander Bakkes, Istvan Szita, and Pieter Spronck. "Monte-Carlo Tree Search: A New Framework for Game AI." Paper presented at the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, California, 22-24 October 2008. <https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-036.pdf>
2. Dickins, Anthony. *A Guide to Fairy Chess*. New York: Dover Publications Inc., 1971.
3. Hartikka, Lauri. "A step-by-step guide to building a simple chess AI." Free Code Camp. Published 30 March 2017. <https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977>
4. Kennedy, Patrick. "Case Study on the Google TPU and GDDR5 from Hot Chips 29." Serve the Home. Published 22 August 2017. <https://www.servethehome.com/case-study-google-tpu-gddr5-hot-chips-29>
5. Lanctot, Marc, Abdallah Saffidine, Joel Veness, Christopher Archibald, Mark H.M. Winands. "Monte Carlo *- Minimax Search." Paper presented at the 23rd International Joint Conference on Artificial Intelligence, Beijing International Convention Center, Beijing, 6-9 August 2013. <https://arxiv.org/pdf/1304.6057.pdf>

6. Pritchard, David. *The Classified Encyclopedia of Chess Variants*. n.p.: John Beasley, 2007
7. Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, et al. "A general reinforcement learning algorithm that masters chess, shogi and Go through self-play." *Science* 362, no. 6419 (2018): 1140-1144.
<https://science.sciencemag.org/content/362/6419/1140>