Creating an AI to play Three Dimensional Chess

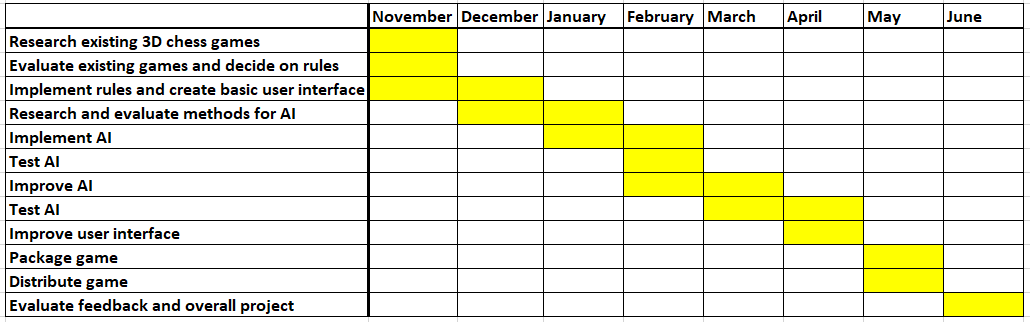
**Finished artefact**

* AppLink
* VideoLink

**Introduction:**

…

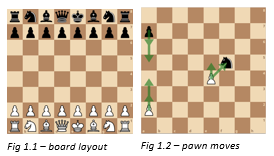
I split the project up into the key tasks which I thought would need to be completed and created a Gantt chart to plan the timings of the process:

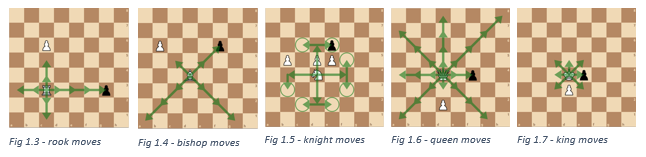


Overall, I stuck to my schedule, but I ended up implementing the AI in December and January and being slightly ahead of schedule as I had some extra time in the holidays.

**2D chess rules:**

My 3D chess game will try to extend the original rules of 2D chess into the third dimension, so to show what I need to extend on, here is a quick summary of the rules of chess:

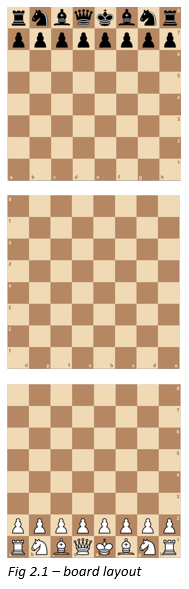
The game is played on an 8x8 board, with the pieces beginning in the layout shown in Fig 1.1. One player controls the white pieces and one controls the black pieces. The players take it in turn to move a piece according to its specific rules, the winner is effectively the player who takes the opposing player’s King. The movement of the pawn is shown in Fig 1.2, it moves one square towards the opposing player’s side. It can also move 2 squares if it is in its starting position, and takes a piece only by moving 1 square forward and one to the side. If the pawn reaches the end of the board it must be swapped for any other piece. The rook moves as far as its line of sight in 1 direction, as in Fig 1.3, and the bishop does the same diagonally i.e. in 2 directions at once (Fig 1.4). The knight hops by a (2, 1) vector, i.e. it moves 2 squares in 1 direction and then 1 square in another direction, hopping over pieces if necessary (Fig 1.5). A queen combines a rook and bishop move (Fig 1.6) and a king does the same but moves 1 square at a time (Fig 1.7).



**Researching and evaluating 3D chess rules:**

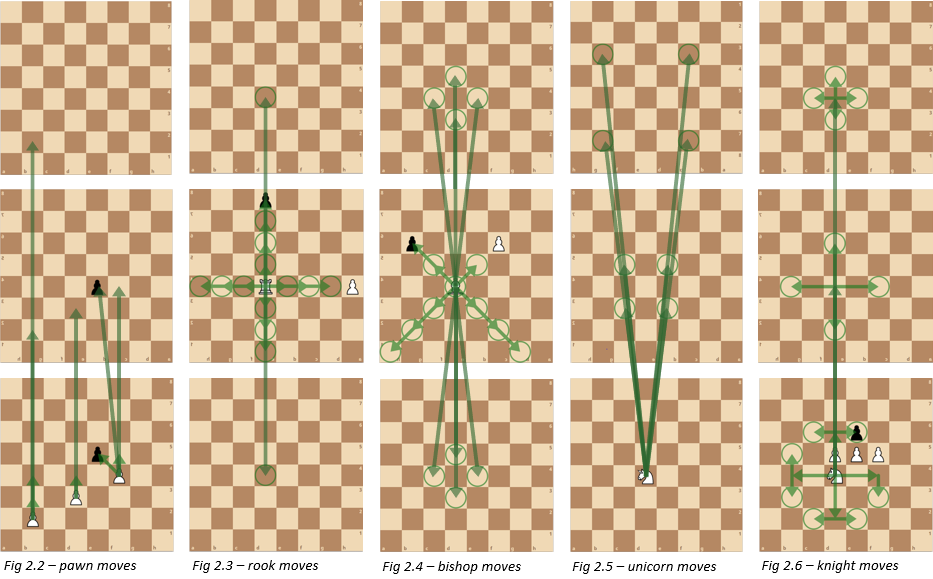
My first step was to decide on the rules my game, as 3D chess has no official rule set or any documented matches. Many people throughout history have theorized and suggested different formats, so I researched the various games in order discover the possibilities for rules. My aim for the rules was to make the game as intuitive as possible as an extension of regular chess, so rules should be similar wherever possible, while also keeping a balanced game which encourages attacking play to a similar extent as regular chess, and does not result in too many draws. I made a table of each game I researched, the specific rules of that game, and my evaluation of those rules.

|  |  |  |
| --- | --- | --- |
| **Version – Creator, Year** | **Notable rules** | **Evaluation** |
| Kubikschack – Kieseritzky, 1851[[1]](#footnote-1) | * First recorded mention of 3D chess * No documented rules, but used 8x8x8 board | - |
| Johnson’s Three-Dimensional Chess – Rick Johnson, 1966[[2]](#footnote-2)  Chess in the Third Dimension – Skor-mor, 1976[[3]](#footnote-3)  Strato Chess – Dynamic Games, 1973 | * 8x8x3 board * Each move consists of a piece moving in its normal way within its own layer, and then moving 1 square either up or down | While this approach is simple and intuitive, I don’t think this is truly a 3-Dimensional game as there is no variety to how the pieces can move between layers. It greatly limits the strategic possibilities and the extra squares with very little extra mobility means there will be fewer piece takes and so a less interesting game |
| Hagemann’s Three-Dimensional Chess – Wally Hagemann, unknown[[4]](#footnote-4) | * Knights move by vector (2, 1, 1) * Rooks move in 1 direction * All other pieces move in a similar way to Johnson’s | The (2, 1, 1) vector movement of the knight seems an unnatural extension of the (2, 1) move in 2D. I like this rook move as it is loyal to the function of rook in regular chess while providing a truly 3 dimensional aspect |
| Raumschach – Ferdiand Maack, 1907[[5]](#footnote-5) | * Most widely played version ever * 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 at once, 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 | Maack’s choice of bishop, rook and knight moves seems to me to best embrace the three-dimensional aspect of the game while remaining as intuitive as possible. I also like the possibility of the unicorn as it is an interesting piece unique to the third dimension but I think the queen should be able to move in only two directions as three gives it too much power. By extension, this leads to a queen-king checkmate, one of the most common in the game, being impossible, so the king should also move in only two directions. Finally, while the 5x5x5 board gives a satisfying cubic shape with a reasonable board size, I think the unusual setup over two layers makes too different to regular chess |



While an 8x8x8 board initially seems ideal, it leads to a strange and unnatural opening, and to quote Pritchard, the 8x8x8 board is “the most mentally indigestible for the players ... Less demanding on spatial vision, and hence more practical, are those games confined to three 8×8 boards and games with boards smaller than 8×8”[[6]](#footnote-6), so I will use an 8x8x3 board with the layout in Fig 2.1 to make it as simple as possible. I settled on the following rules:

The pawn moves forward, either horizontally or vertically, and takes only by moving 1 square forward and 1 to the side (Fig 2.2). The rook moves in 1 direction (Fig 2.3), the bishop moves in 2 directions (Fig 2.4) and the unicorn, an optional replacement for knights, moves in 3 directions (Fig 2.5). The knight hops by a (2, 1, 0) vector (Fig 2.6), a queen combines a rook and bishop, and a king does the same but 1 square at a time. I then implemented these rules into an initial python app.

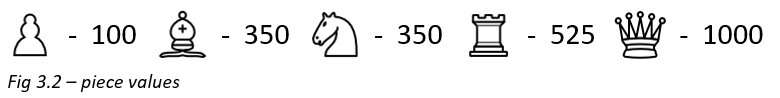


**Researching and evaluating AI:**

I then began to look at the possible methods of creating an AI and evaluated each one:

|  |  |  |
| --- | --- | --- |
| **AI method** | **Summary** | **Evaluation** |
| Deep learning from match database | The usual approach to all chess AIs and engines, data from thousands of matches is analyzed to teach a program which moves are beneficial | While this would normally be the obvious choice due to its efficiency and skill, but since 3D chess is not widely played, there are no games to analyze so this is not possible |
| Deep learning from first principles[[7]](#footnote-7) | An program is given the rules of the game and plays millions of matches against itself to learn the best ways to win. | “5000 first-generation … and 16 second-generation TPUs were used to train the neural networks … for approximately 9 hours.”[[8]](#footnote-8)  With the 179600GB/s bandwidth used here[[9]](#footnote-9) compared to my available 40GB/s and using the data from Fig 3.1[[10]](#footnote-10), it would take me 1150 hours of continuous computing power to reach the level of a casual player, so I will have to use another method |
| Monte Carlo Tree Search[[11]](#footnote-11) | The algorithm uses randomized explorations to explore a tree of all possible moves and decide on the best one | Uses 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]](#footnote-12) | “Designed for … games where one would rarely expect to sample the same successor state multiple times”[[13]](#footnote-13) – this is ideal as positions are rarely repeated in chess. Has a large skill gain with little computing power, especially with Alpha-Beta pruning, so I will use this algorithm |

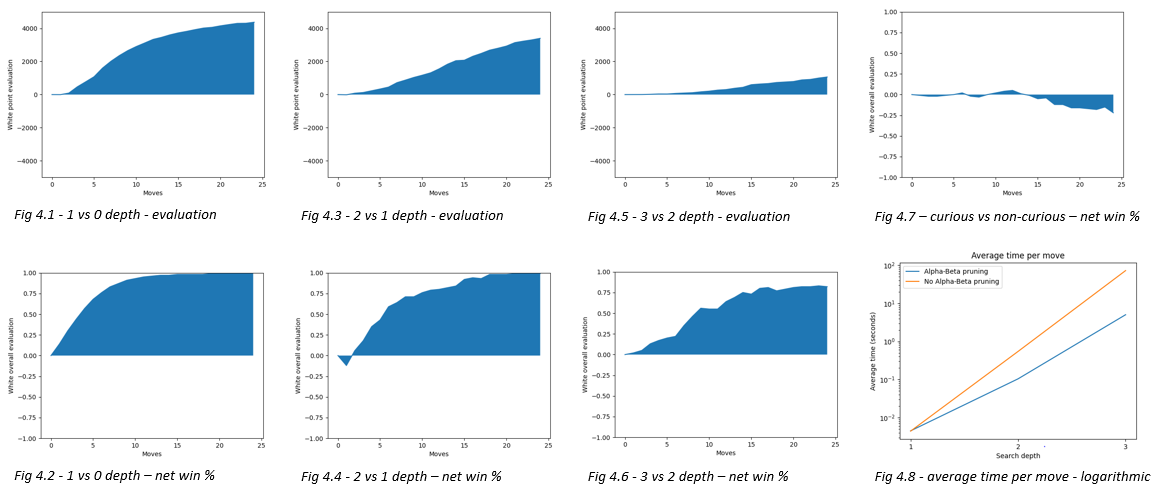
I then implemented the Minimax Search in python[[14]](#footnote-14), adding up the values of the pieces on the board as the evaluation function. I initially used the values from 2D chess (Fig 3.2).



**Testing and evaluating the AI:**

I then carried out some tests on 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, and the search depth is clearly vital to the skill of the algorithm (Fig 4.1-4.6), so I need to run the program at as high search depth 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 a beneficial.

Next, I tested the time taken by the program 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. I think 1 second is a sensible time per, so currently the highest search depth possible is 2.



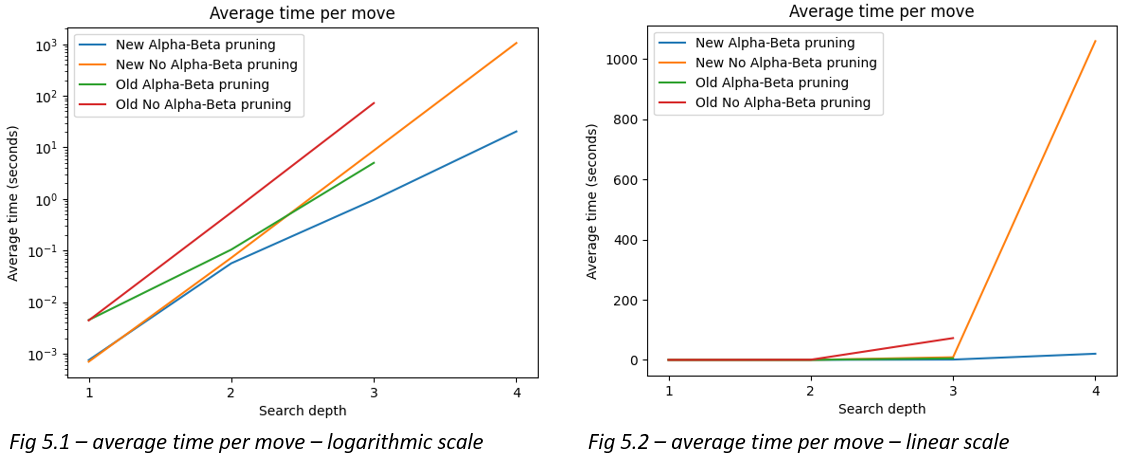
Finally, I played 10 games against the AI at 2 depth. In the first few games, I found it difficult to visualize the moves in 3D and so the AI capitalized on my mistakes and won comfortably. As I got used to the movements, it began to be more even and the game became more strategic, but while the AI was 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. From these games I think the way to best improve the AI would be to increase the search depth and give the AI some more positional awareness, as the moves were effectively random when there were no available piece takes.

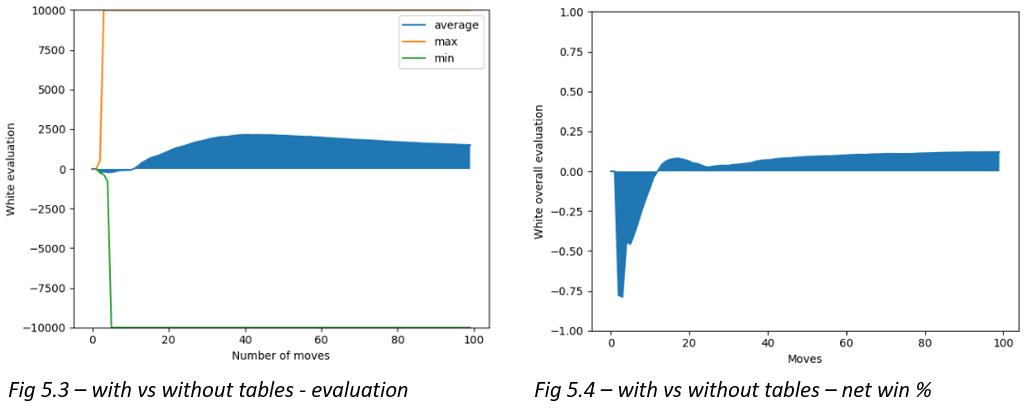
**Improving the AI:**

I improved the positional awareness by creating ‘piece-square tables’ for each piece, 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 dimenson. I also sped up the program by only evaluating the board at the start of the minimax and then adding on the change in value of the piece which is moving.

**Testing and evaluating the AI:**

To test the effectiveness of my optimization, I created more graphs of the time taken for the AI at different search depths for the old and new algorithms, with and without Alpha-Beta pruning (Fig. 5.1-5.2). The change seems to have a significant effect on timings at all levels, and crucially, it brings the 3 move time below 1 second, allowing me to use that as my chosen depth. I also played 10000 games of tables against no tables (Fig 5.3-5.4), and it is clear these are marginally advantageous, although only after the 10th move.





**Finishing the product and feedback:**

I added some extra settings and features to make the app more user-friendly and give the option to experiment with the different rules such as the number of pawn rows and whether to use knights or unicorns, and packaged it into a final product.

Finally, I played my AI against some highly ranked chess players, losing narrowly to a 1900 rated player and beating a 1700 player. From this I estimate the skill level of the AI at 1800ELO, ranking it as a Class A player, one rank off Candidate Master.

**Conclusion and evaluation:**

…

1. Anthony Dickins, *A Guide to Fairy Chess*, (New York: Dover Publications Inc., 1971), 16-17 [↑](#footnote-ref-1)
2. David Pritchard, *The Classified Encyclopedia of Chess Variants*, (John Beasley, 2007), 225-233 [↑](#footnote-ref-2)
3. Ibid. [↑](#footnote-ref-3)
4. Ibid. [↑](#footnote-ref-4)
5. Dickins, *A Guide to Fairy Chess,* 16-17 [↑](#footnote-ref-5)
6. Pritchard, *The Classified Encyclopedia of Chess Variants*, 305 [↑](#footnote-ref-6)
7. David Silver, “A general reinforcement learning algorithm that masters chess, shogi and go through self-play”, ScienceMag.org, Science *362,* no. 6419 (December 2018): 1140-1144 [↑](#footnote-ref-7)
8. Silver, “A general reinforcement learning algorithm”, 1142 [↑](#footnote-ref-8)
9. Patrick Kennedy, “Case Study on the Google TPU and GDDR5 from Hot Chips 29”, servethehome.com, published August 22, 2017, https://www.servethehome.com/case-study-google-tpu-gddr5-hot-chips-29/ [↑](#footnote-ref-9)
10. Silver, “A general reinforcement learning algorithm”, 1140 [↑](#footnote-ref-10)
11. Guillaume Chaslot, “Monte-Carlo Tree Search: A New Framework for Game AI” (paper presented at the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, California, October 22-24 2008), https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-036.pdf [↑](#footnote-ref-11)
12. Marc Lanctot, “Monte Carlo \*- Minimax Search” (paper presented at the 23rd International Joint Conference on Artificial Intelligence, Beijing International Convention Center, Beijing, August 6-9 2013), https://arxiv.org/pdf/1304.6057.pdf [↑](#footnote-ref-12)
13. Ibid. [↑](#footnote-ref-13)
14. Lauri Hartikka, “A step-by-step guide to building a simple chess AI”, freecodecamp.org, published March 30, 2017, https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/ [↑](#footnote-ref-14)