Creating an AI to play Three Dimensional Chess

**This project develops the chess game into the third dimension, first as a 2-player game and then adding an AI layer that you can play against.**

**[RM: I think it would read better if you summarise the challenge in 2 lines here before you start anything else. You can move the text from below, so something like……**

**[RM: normally you’d right these reports in the 3rd person not 1st person. Do you know what school expects?]**

**[RM: I think you could abbreviate the references. Are you allowed to?]**

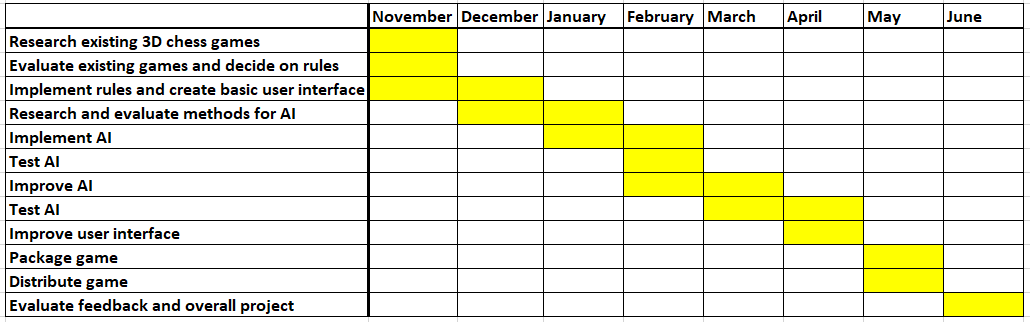
**Finished artefact**

* 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:

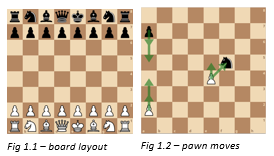


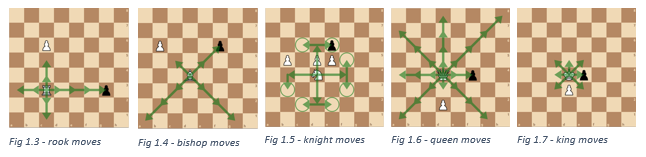
Overall, I kept to schedule, in fact implementing the AI one month early due to extra time in the holidays.

**Quick recap: 2D chess rules:**

[RM: safe assumption that the reader understand chess but I think you can cheat on the wordcount by adding an image.]

The rules of 2D chess on an 8x8 board with white against black are well known, with permitted moves as summarised below:





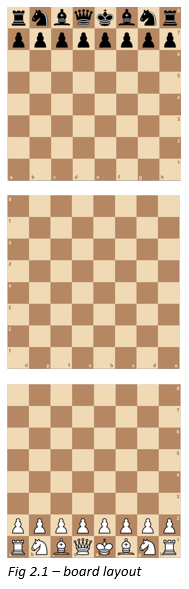


**Phase 1: Researching and evaluating rules:**

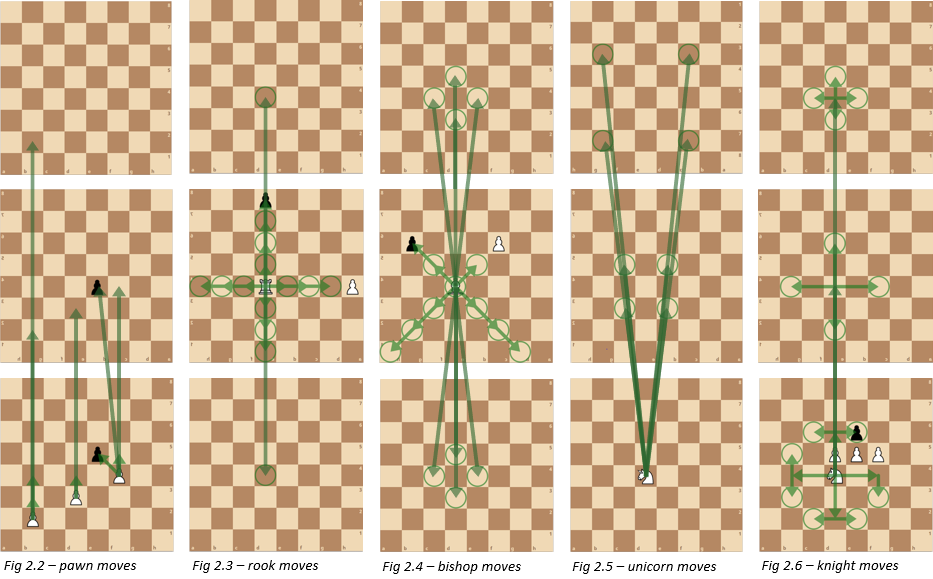
As 3D chess has no official rule set or any documented matches, the rules must be decided. Many people throughout history have theorized and suggested different formats, so I researched the various games in order discover the possibilities for rules. My aim for the rules was to make the game as intuitive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

Table 1 [RM: I think you could reduce word count in the table, although it’s interesting so not too much. Maybe change column 3 to bullets?]

|  |  |  |
| --- | --- | --- |
| **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 * Pieces move normallyplus 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  [Bullets? For example….]   * 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]](#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 * Moves evolved from 2D to 3D for all pieces eg knights move in a (2,1,0) vector across any 3 dimensions | 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. The optinoal unicorn is interesting as a unique addition to the third dimension. However I think the queen should be limited to two directions to avoid it having too much power and the kind should be limited to two directions to ensure a queen-king checkmate is possible. While the 5x5x5 board gives a satisfying cube with a reasonable board size, I think the unusual setup over two layers 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... 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). I therefore decided to adopt an 8x8x3 board with the layout in Fig 2.1 to make it as simple as possible. I settled on the rules illustrated below: [RM: could you delete this paragraph and rely solely on the images? Or leave one written explanation just as an example]



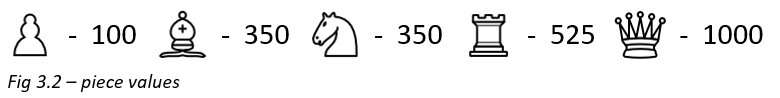
**Phase 2: Researching AI options:**

With rules agreed, the next phase was to examine and evaluate 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]](#footnote-7) | A 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  [Note sure the distinction between MCTD and MCMS is clear. Does MCTS look at moves to check-mate whereas MCMS stops before then?] | 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 [RM: I wonder if you should define terms like backpropagation?] |
| 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 |

[RM: worth a comment on why you decided on this point scoring evaluation function?]

To implement the Minimax Search, it is necessary to develop an ‘evaluation function’ that gives a point score to any arrangement of pieces on the board. Based on successful strategies in 2D chess[[14]](#footnote-14), I chose a function that assigns the following scores to each piece and sums these to give a score for each player (Fig 3.2).



[RM: out of interest, did you try changing the point system?]

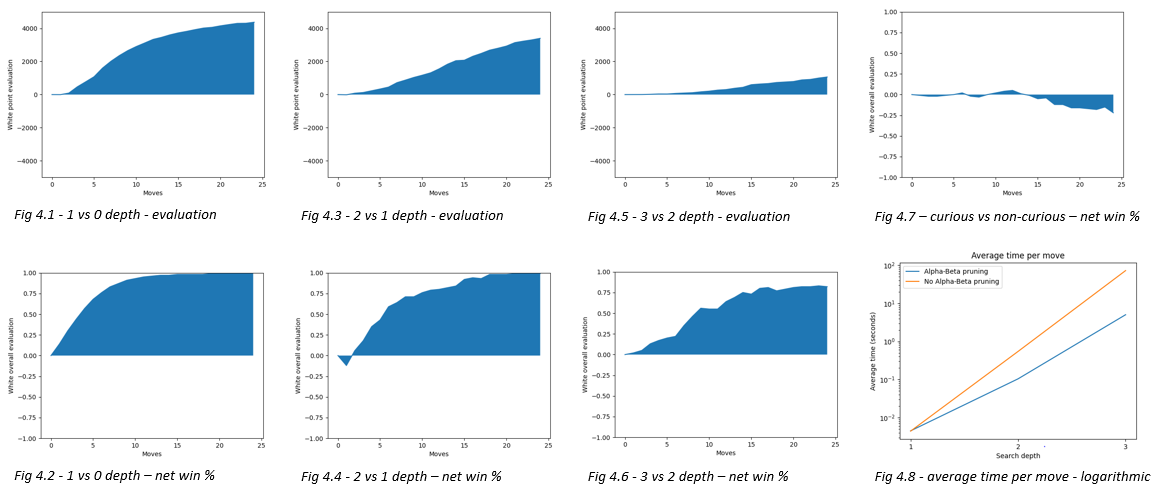
**Phase 3: Testing and evaluating the AI:**

In this phase, I testes 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.

For quality game-play, I decided on a target of 1 second per move, limiting the highest search depth possible to 2.

[RM: This is a key conclusion and very interesting. It might be worth explaining a bit more as it’s not simple to understand how you get to your conlusion from the graph]



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.

Based on these games, I concluded that the best optimisation strategy would be increasing 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’) 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.

[I wondered if you could make this clearer and fewer words in a table……]

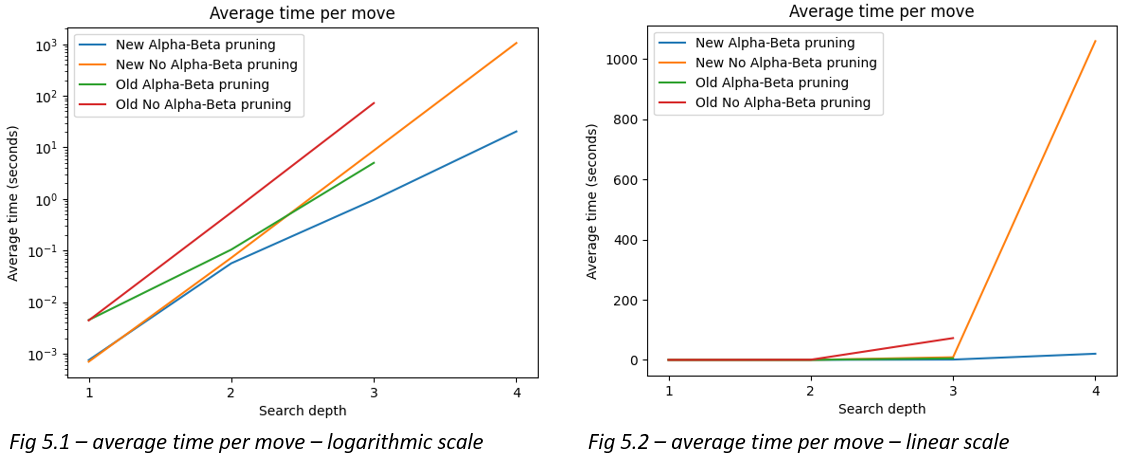
I then carried-out the following optimization tests:

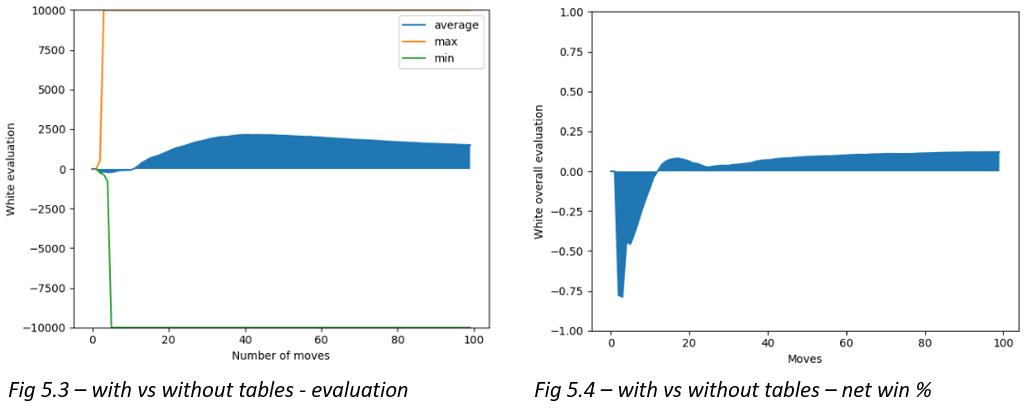
|  |  |
| --- | --- |
| Test | Conclusion |
| Old v new algorithm, with and without ABP, search depths 1-4 | Move conclusion here |
| 10,000 games, PST v no PST | Move conclusion here |

[RM: I got lost a bit here. What’s the old and new algorithm if not PSTs or alpha-beta pruning?]

Then, 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 had a significant effect on timings at all levels, and crucially brought the 3 move time below 1 second, allowing me to use that as my chosen depth.

Playing 10000 games of the PST algorithm against no original (Fig 5.3-5.4), and it is clear these are marginally advantageous, although only after the 10th move.





**Phase 5: Player feedback:**

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

Finally, the AI was played 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 below Candidate Master.

[RM: Do you have player a quote you could add?]

**Conclusion:**

…[RM: Where is it?]

[Move video and app link to conclusion?]

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)