Creating an AI to play Three Dimensional Chess

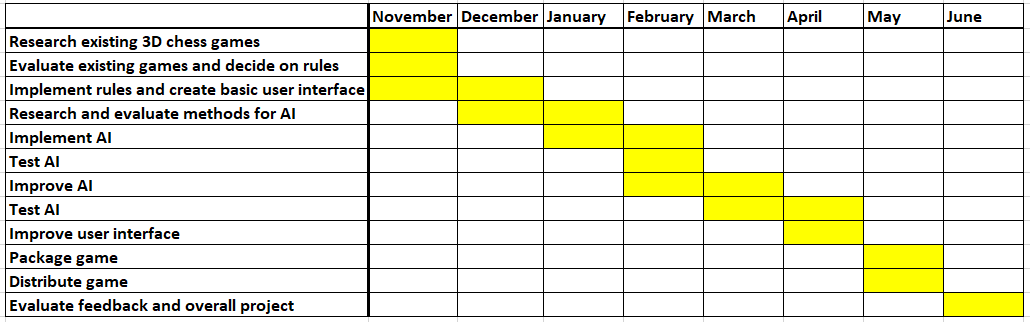
**Finished artefact**

* Link to app
* Link to video playing against a person

**Introduction:**

…

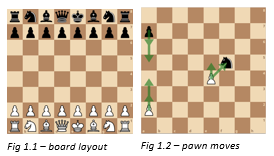
I split the project up into the key tasks which I thought would need to be completed and created a Gantt chart to plan the timings of the process:

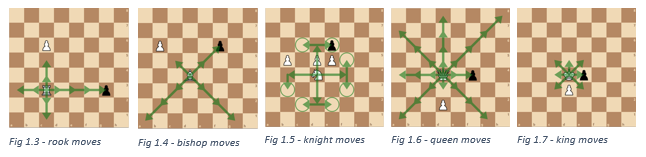


Overall, I stuck to my schedule, but I ended up implementing the AI in December and January and being slightly ahead of schedule as I had some extra time in the holidays.

**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:

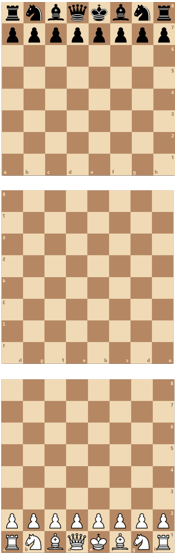
The game is played on an 8x8 board, with the pieces beginning in the layout shown in Fig 1.1. One player controls the white pieces and one controls the black pieces. The players take it in turn to move a piece according to its specific rules, the winner is effectively the player who takes the opposing 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:**

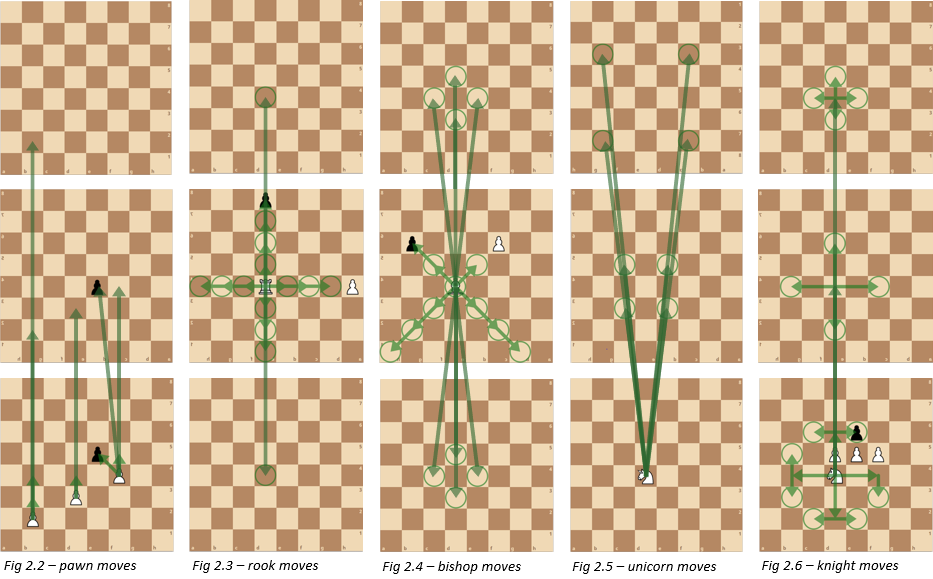
My first step was to decide on the rules my game, as 3D chess has no official rule set or any documented matches. Many people throughout history have theorized and suggested different formats, so I researched the various games in order discover the possibilities for rules. My aim for the rules was to 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 |
| Space Chess – Pacific Games Co., 1969[[4]](#footnote-4) | * Exceptionally complicated rules which defy simple description | I need to make sure my game is intuitive enough to at least by taught to a casual chess player by a short decription |
| Hagemann’s Three-Dimensional Chess – Wally Hagemann, unknown[[5]](#footnote-5) | * 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[[6]](#footnote-6) | * Most widely played version ever * Began with 8x8x8 board * Experimented with 7x7x7 board, with 2 extra pieces: the giraffe, which moves by the vector (4, 1, 0) * 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 an interesting piece unique to the third dimension, but the giraffe seems unnecessary. I think the queen being able to move in all three directions gives it too much power over the other pieces, and being a combination of a rook and bishop is a core property, so I think it should only be able to move in two directions. By extension, this movement leads to a checkmate with just a queen and king, one of the most common chekmates in the game, being impossible, so the king should have the same movement but only one square at a time. 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 for casual chess players and removes the possibility for the double pawn move which is an integral part of chess |

While an 8x8x8 board initially seems ideal, it leads to a strange and unnatural opening as the black queen can be taken diagonally on the first move by the white queen, which should not be a viable beginning to a game. Pritchard said that the 8x8x8 board is “the most popular 3‑D board amongst inventors, and at the same time 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”2, and I think that the 8x8 layers are essential to keep the familiar layout and make the game more accessible, so I will use an 8x8x3 board to make it as easy as possible to represent on a screen and visualize for a player. After researching all the past versions and exploring the possible rules, I decided on the following set of rules:

The game is played on an 8x8x3 board, with the pieces beginning in the layout shown in Fig 2.1. The knights may also be replaced with unicorns. The pawn moves 1 square forward, either horizontally or vertically, or 2 squares if it is in its initial position. It takes only by moving 1 square forward and 1 to the side (Fig 2.2). The rook 1 direction (Fig 2.3), the bishop moves in 2 directions i.e. in two-dimensional diagonals (Fig 2.4) and the unicorn 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 moves 1 square at a time. I then implemented these rules into an initial python app.

Fig 2.1 - board layout



**Researching and evaluating AI:**

I then began to look at the possible methods of creating an AI to play my game and evaluated ones pros and cons specific to the task in a table:

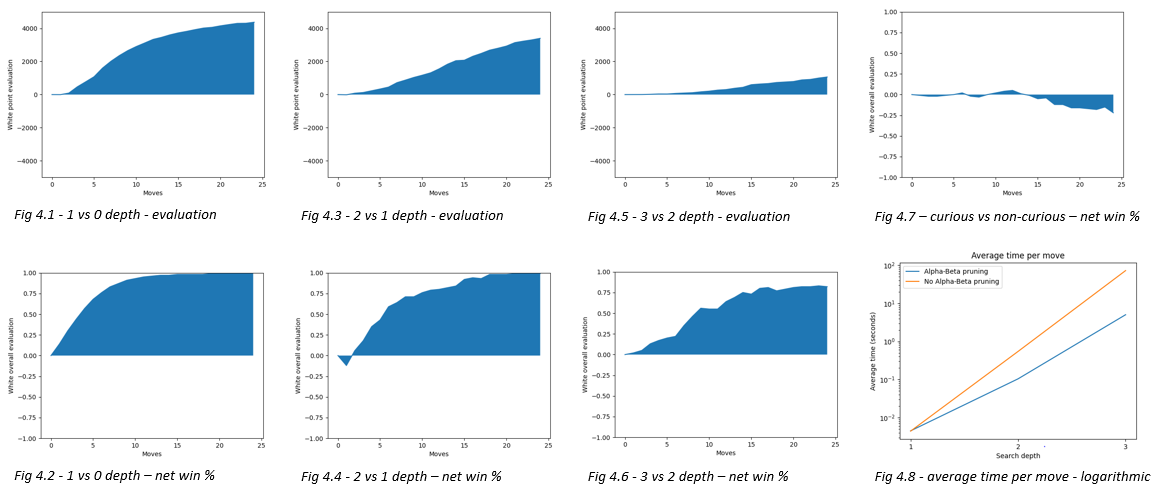
|  |  |  |
| --- | --- | --- |
| **AI method** | **Summary** | **Evaluation** |
| Deep learning from match database | The usual approach to all chess Ais and engines, data from thousands of matches is analyzed to teach an ‘intelligent’ program which moves are beneficial in each scenario | While this 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 ‘intelligent’ program is given only the rules of the game and plays millions of matches against itself to learn the best ways to win. Has been successfully used in the past to create very powerful chess engines | “During training only, 5000 first-generation tensor processing units used to generate self-play games, and 16 second-generation TPUs were used to train the neural networks. Training lasted for approximately 9 hours in chess.”[[8]](#footnote-8)  The total bandwidth of 5000 1Gen and 16 2Gen TPUs is 179600GB/s[[9]](#footnote-9), while the computers I have access to have approximately 40GB/s. With this difference in processing power and the data from Fig 3.1[[10]](#footnote-10), it would take me 1150 hours or 48 days of continuous computing power just to reach 1000ELO, roughly the skill level of a casual player. This is impractical for my project, so I will have to use another method |
| Monte Carlo Tree Search[[11]](#footnote-11) | A tree of all the possible sequences of moves is created and the algorithm uses randomized explorations to decide on the best move | Uses a method of backpropagation to store the value of 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 assuming the opponent makes perfect moves, and makes the move with the best evaluation. It can be optimised with alpha-beta pruning to speed up the process and allow a deeper search[[12]](#footnote-12) | “The algorithm is designed for the class of densely stochastic games; that is, games where one would rarely expect to sample the same successor state multiple times at any particular chance node”[[13]](#footnote-13) – this method is ideal as positions are very rarely repeated in chess. The alpha-beta pruning also improves it further, allowing a relatively large gain with little computing power, and the whole algorithm is relatively easy to implement. |

Having researched the possibilities for my program, I decided on a Monte Carlo Minimax Search with Alpha-Beta Pruning for optimization, as it is the best fit for my project. I implemented the algorithm in python[[14]](#footnote-14), adapting it for 3D chess. The program requires an ‘evaluation function’, an algorithm to decide the value of a board position, effectively deciding which player is winning. To begin I simply added the values of the pieces each player has, using the values from 2D chess (100 for a Pawn, 350 for a Knight or Bishop, 500 for a rook, 1000 for a Queen and 10000 for a King), and I think this should represent the game fairly well as I tried to make the pieces as similar in value as possible.

**Testing and evaluating the AI:**

I then carried out a number of tests on the AI to analyze its performance and skill level. Firstly, I let the AI play against itself at varying depths to compare its abilities, and the search depth is clearly vital to the skill of the algorithm with each depth being ahead in half the games no later than the 5th move and 80% before the 25th move when playing the depth below. It is clear from this that I need to run the program at as high search depth as possible to gain maximum ability. 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 strategy as it leads to a more conservative style.

I then tested the time taken by the program to decide a move at different search depths (looking a different number of moves ahead), 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, but I think 1 second is a sensible time per move so the user doesn’t have to wait too long between moves, so currently the highest search depth possible is 2.



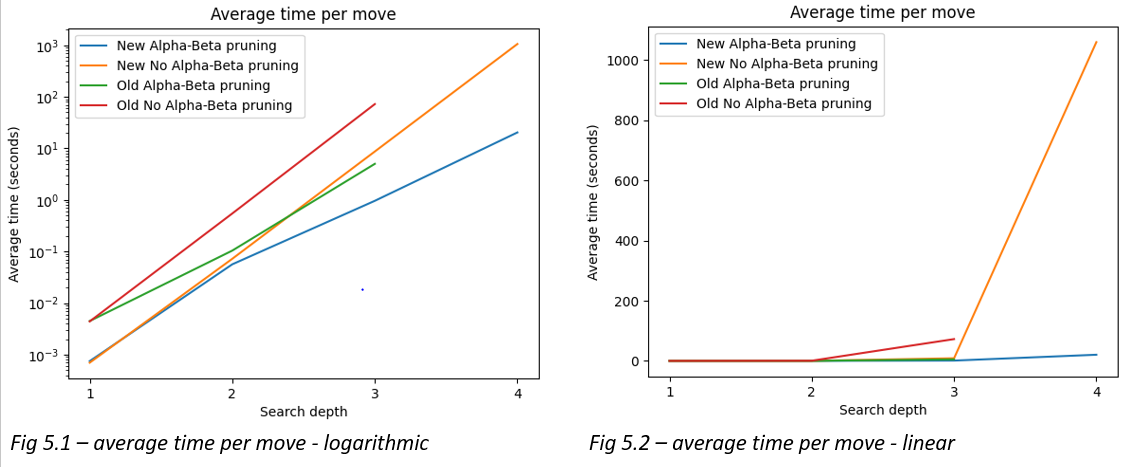
Finally, I played 10 games against the AI at 2 depth. In the first few games, I found it very difficult to visualize the moves in three dimensions and so the AI capitalized on my mistakes and won comfortably. Then as I got used to the piece movements, it began to become much more even and the game became more strategic, but while the AI was still ahead, it struggled to win the game as it simply couldn’t force a checkmate. In the final games the same effect was reversed, as I found it hard to win on a very open board, but I only had a marginal piece advantage. 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 when there were no takes available it was effectively taking random moves which did not gain any advantage.

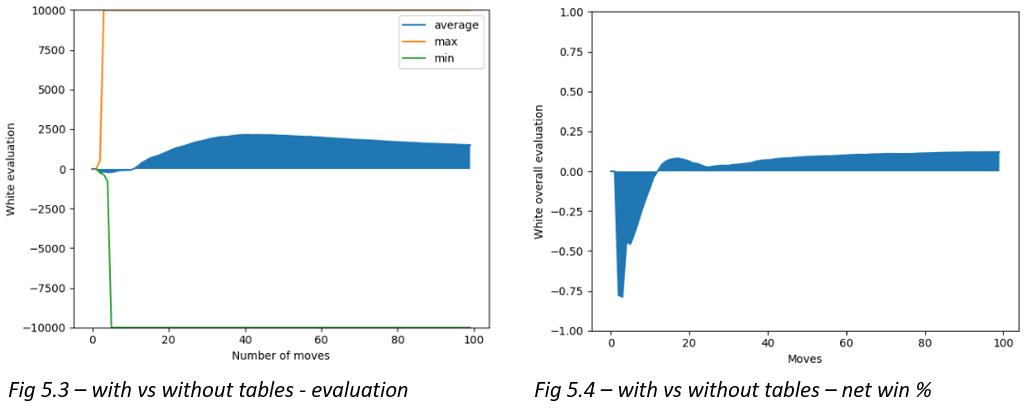
**Improving the AI:**

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, with the old and new algorithms with and without alpha-beta pruning on a logarithmic and linear scale (Fig. 5.1, 5.2). The change increases speed 10 times at low depths, and seems to have a significant effect on timings at all levels. Crucially, it brings the 3 move time below 1 second to 0.96, allowing me to use that as my chosen depth. I also played 10000 games with white using tables and black without them on 1 depth (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.

I got a few chess players to play the game against my AI, and the games were very close. First I played against a competitive tournament player rated 1900, and while he said the computer played well and challenged him, the computer narrowly lost. I then played against a younger player rated 1700, and the AI was always on top, winning after 82 moves. Based off these games, I estimate the skill level at 1800ELO, making my program a Class A player, 1 rank off Candidate Master. They also said the program was easy to use, but suggested a couple of improvements which I then implemented.

**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. Ibid. [↑](#footnote-ref-5)
6. Dickins, *A Guide to Fairy Chess,* 16-17 [↑](#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)