IS CHESS PREDICTABLE?

By Sulaiman Barzangi

At first glance, chess appears to be a perfectly logical game. Every position follows a fixed set of rules, each decision carries clear consequences, and nothing is left to chance. In principle, a game of chess should be entirely predictable, with its outcome being determined by calculation from the starting position.

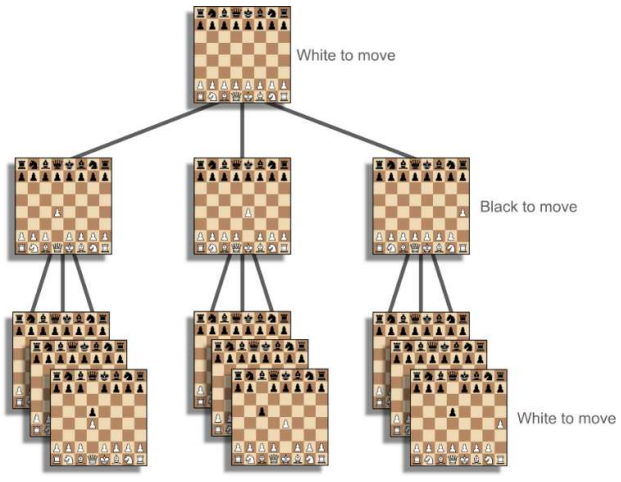
However, even the strongest chess players, and the most advanced chess engines, are unable to predict chess games from the beginning. The sheer number of games possible in chess, estimated by Shannon's Number (approximately 10^{120}), is so vast that it becomes impossible to analyze. As a result, chess remains highly unpredictable, despite containing no randomness at all.

This raises a deeper question: Can a deterministic system, like chess, be considered predictable if it cannot be computed in practice?

Before proceeding, the word 'predictable' can be distinguished between two meanings. One in the theoretical sense and the other in the practical sense. Chess is predictable in the theoretical sense because its rules determine every outcome. In the practical sense, however, it means that it is computable within physical limits. This essay focuses on the latter.

Imagine this: an intergalactic being with infinite computation power, an alien who could compute all 10^{120} games in its head in an instant. For that alien, chess would be perfectly predictable. But no physical computer, now or ever, can do so. Chess is only predictable in the hypothetical field and not in the physical one.

To understand why chess is difficult to predict, it is useful to consider that each chess position offers many different moves, and after each move, more pathways branch out. This forms a structure known as a game tree. On average, a single position in chess has around 30 to 35 legal moves available, and after each move, a new position is formed with its own new set of possibilities. This increases the number of possible positions exponentially with each move played. For instance, after just a small number of moves, the total number of possible positions already reaches into the billions. This rapid growth in numbers makes it impractical to analyse every possible continuation, which supports the idea that although chess is deterministic, it cannot be fully predicted in practice.



Initially, the model to identify the number of possible games was thought out to be approximately

$$N \approx a^{2n}$$

Where N is the number of possible chess games, a is the number of available legal moves (the branching factor), and n is the number of moves (2n is required because each move contains 2 plies and each ply contributes to the branching factor)

When Shannon used this model to find the total number of possible chess positions, he assumed the value of a to be a constant between 30 and 35.

So, to model number of probabilities 'N', we get the following exponential relationship:

Moves (n)	Estimated Number of Possibilities (N where $a = 35$)
1	1.23*10^3
2	1.50*10^6
3	1.84*10^9
4	2.25*10^12
5	2.76*10^15
6	3.38*10^18
7	4.14*10^21
8	5.08*10^24
9	6.21*10^27
10	7.61*10^30
11	9.32*10^33
12	1.14*10^37
13	1.40*10^40
14	1.71*10^43
15	2.10*10^46
16	2.57*10^49
17	3.15*10^52
18	3.86*10^55
19	4.73*10^58
20	5.79*10^61

Chess TMR | Desmos

After only 3 moves, the number of possible outcomes is already in the billions, and after 52 moves, that number would have surpassed 10^{80}, which is approximately the number of particles in the observable universe, according to Arthur Eddington.

However, this model for 'N' is overly simplistic as the number of legal moves available 'a' differs constantly, based on the position and the number of pieces left on the board, for example: at the starting position, the number of legal moves available is 20, but in positions where a high value piece is in the center, it can control a lot more of the squares on the board and raise that number above the 30s. Branching factor 'a' is not a constant, hence the simple model only gives a rough estimate and ignores how chess behaves.

You could achieve the same chess position with different games. The number of possible games corresponds to the number of distinct paths throughout the game tree, while the number of positions corresponds to the number of nodes. Since multiple paths can lead to the same position, the number of games is significantly larger than the number of positions.

Shannon's number remains a useful estimate, but both points; the variability of the branching factor 'a' and the distinction between games and positions, highlight the complexity of chess.

A more refined model would be a model that records all possible games, with a set value a for each position:

$$N = a_1 * a_2 * a_3 * \dots * a_{2n}$$

$$log_{10}(N) = log_{10}(a_1 * a_2 * a_3 * \dots * a_{2n})$$

$$log_{10}(N) = log_{10}(a_1) + log_{10}(a_2) + log_{10}(a_3) + \cdots + log_{10}(a_{2n})$$

$$log_{10}(N) = \sum_{i=1}^{2n} log_{10}(a_i)$$

This formulation allows for the branching factor to be implicated into the model. Although a_i differs throughout the game, their average remains sufficient that the sum grows approximately linearly with the number of moves*:*

$$log_{10}(N) \approx 2n * log_{10}(\bar{a})$$

where $\bar{a}$ is the average branching factor (between 30-35). Since $log_{10}(\bar{a})$ is a positive constant, $log_{10}(N)$ linearly increases as n increases; therefore, N itself increases exponentially with n. This demonstrates that even a small increase in moves results in a massive increase in possible games, reinforcing the idea that complete computation is infeasible.

Because the different values for a_i varies a lot in a way where we can't plug all the values in, the refined model does not aim to produce a fundamentally different estimate to the simple model, but rather to justify the exponential model by accounting for variability in the branching factor.

If there are 10^{120}(Shannon's Number) possible games, a computer could not compute that before the time Earth gets destroyed by the sun, or the heat death of the universe. For instance, if the fastest supercomputer on earth, El Capitan, manages to analyze 10^{18} or a quintillion game possibilities a second, it would still take over 10^{94} years to fully compute. To put that into perspective, if you had started this calculation since the start of the big bang, you would only be about:

0.00000000000000000000000000000000000000000000000000000000000000000000000000000001% complete.

Similarly, Shannon explained this in his paper (March 1950) that predicting chess with brute-force is impossible. So, an alternative way to predict or solve chess, is to simplify chess itself. Instead of considering every game possibility, considering only logical games is a more feasible approach, which is what chess engines today do to compute the best moves. Instead of branching the game tree into ~35 branches each turn, only take the best moves available in the position, this is called a Heuristic Evaluation. Claude Shannon used a heuristic evaluation based on an evaluation function that estimates a position for optimal long-term advantages based on the following features:

- Material Advantage.
- Pawn Formation.
    - Backward, isolated, and doubled pawns.
    - Relative control over the centre (pawns at e4, d4, c4).
    - Weakness of pawns near king (e.g., advanced g pawn).
    - Pawns on opposite colour squares from bishop.
    - Passed Pawns.

- Position of Pieces.
    - King Safety
    - Control over the Centre.
    - Advanced Knights.
    - Rook on Open File, or Semi-Open File.
    - Rook on the Seventh Rank.
    - Doubled Rooks.
- Piece Activity.
    - Pieces required to protect another piece, committed with limited mobility
    - Attacks on pieces, giving a player the choice to exchange
    - Attacks on the same square adjacent to the king.
    - Skewers.
    - Pins.
- Mobility.

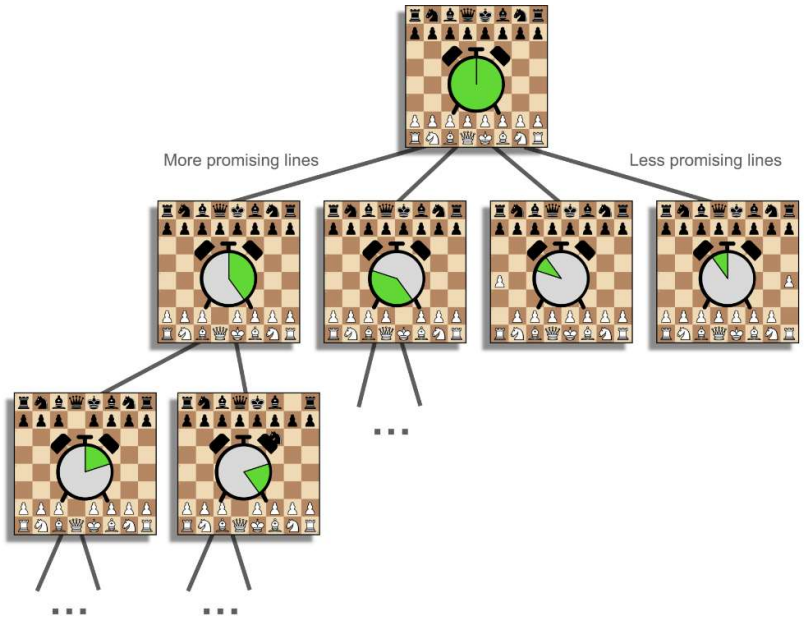
The concept of a Minimax algorithm is that it explores possible moves ahead of the game tree and scores the position based on the before-mentioned evaluation function. It works backwards from the position it wants to reach based on that position's score to the position where it initially is.

For a simple example: Suppose white can capture a knight (3 points), but sees that after that capture, black can take white's rook (5 points). The Minimax algorithm would prioritize preventing the white rook from being captured over capturing the black knight.

Chess engine Stockfish 18 prefers to move the rook to safety (Ra6) than taking the knight on f5 (Bxf5)

Minimax assumes both opponents play optimally where it assumes you play the best move and the opponent minimizes your advantage. By using a technique called Alpha-Beta Pruning, it eliminates branches that do not influence the final decision, which reduced the number of nodes evaluated.

In the example above, Alpha-Beta Pruning would skip evaluating all branches once it realises that white's rook being captured results in a losing score. There would be no need to evaluate other moves if it would be guaranteed to end up worse for white anyways, so less time is spent on evaluating the less promising lines and divides it's time to the more promising lines (see diagram below).



This method is used by modern chess engines when calculating a move to save calculation time and maximise the quality of each chess move by prioritizing certain features in a position. It is also useful to consider that opening moves are studied very thoroughly and therefore is easier to evaluate perfectly. Similarly, an endgame position with limited pieces is also studied very thoroughly and therefore also easier for it to be perfectly evaluated, as supposed to a middle-game position that has never happened in chess before.

A limitation to using these evaluation techniques to compute future chess positions, is that the engine cannot see far enough ahead and might suggest a suboptimal move which only becomes clear many moves later, this is called the horizon effect.

A way chess engines combat the horizon effect is by not relying on certain rules but rather playing many games against themselves to recognise winning patterns, and by using collected data to decide a move, if two decisions are equally appropriate, using deterministic logic. For example, if an opening move as white has a 70%-win rate, and another move has a 65%-win rate, then the engine will likely prioritize playing the first move, even then, the best that chess engines can do is provide a very educated guess to what a good move is.

Although algorithms such as Minimax and Alpha-Beta pruning significantly reduce the complexity of the problem, they do not fully eliminate the uncertainty of chess. Due to the combinatorial explosion of possible positions, limitations in evaluation functions, and restricted search depth, chess remains fundamentally unpredictable beyond certain limits.

A limitation of this essay is that, for simplicity, it does not account for quantum computers, which lie outside the scope of classical computing, and are not addressed here.

In conclusion, chess, as Shannon said himself, cannot be explained using brute force alone, as demonstrated by Claude Shannon's analysis of the immense complexity of chess. While algorithms such as minimax and alpha-

beta pruning reduce the number of positions evaluated, it still cannot evaluate the entire game in one go as they are reliant on heuristics, and for that reason has its limits with the horizon effect and cannot guarantee perfect chess. Although modern chess engines like AlphaZero and Stockfish play at an extremely high level, they only do so through approximation and not by prediction. Ultimately, chess remains a complex system where perfect foresight is computationally unobtainable, serving as a benchmark for future artificial intelligence models and problem-solving algorithms. Some might argue that technology will never reach that level, but Shannon's number isn't just big, it's astronomically beyond physical limits. Chess, then, is a strange kind of infinity: one that is entirely determined, and yet forever beyond our grasp. It reminds us that computation has limits, and that some questions, even when the rules are clear, may never have a single correct answer.

REFERENCES

Allis, L. V. (1994). *Searching for solutions in games and artificial intelligence* (Unpublished doctoral dissertation). University of Limburg. Retrieved from http://fragrieu.free.fr/SearchingForSolutions.pdf

Buttner, C. (2021). *High-level explanation*. ChessCoach. https://chrisbutner.github.io/ChessCoach/high-level-explanation.html

Desmos. (n.d.). *Chess TMR* [Desmos graph]. Retrieved April 8, 2026, from https://www.desmos.com/calculator/0pg7rnrgvv

Shannon, C. E. (1950). *XXII. Programming a Computer for Playing Chess* (Philosophical Magazine, Ser. 7, Vol. 41, No. 314). Bell Telephone Laboratories. https://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf