Monte Carlo Nought and Crosses

Shawn C. Y. Tan

Undergraduate, Physics Department, Imperial College, London SW7 2BB

A study on the game 'Noughts and Crosses' was conducted using Monte Carlo statistical methods, which involved the simulation of randomly played games. Checking for win conditions proved to be the most resource intensive process, and several computational strategies were explored to determine the most efficient. This was determined to be an algorithm utilising vectorised calculations and the binary erosion method from the *Scipy* image processing library. Subsequently, the effect of increasing board size on game length and advantage was investigated. Game length was found to increase non-linearly, while advantage was found to tend to zero.

I. Introduction

The study of games has always been a mathematical fascination, so much so that it spawned a whole discipline: game theory. This discipline has had far reaching consequences, with applications in diverse fields such as economics and computer science [1]. A particular form of board games involving tokens have a long history, with traces from as far back as ancient Egypt [2]. It has evolved into the noughts and crosses game we know today – a two-player strategy game with a certain set of rules. The controlled environment provided by such a game offers a test ground suitable for testing mathematical techniques and computational tools. Adjustments to the rules can be made to gain insight into more complex systems such as quantum ones [3].

Monte Carlo methods are a branch of numerical techniques used to model solutions to complex problems that would prove difficult to do analytically [4]. These methods rely on generating random samples and using statistical sampling to draw conclusions about the population. Monte Carlo methods are used widely across differing areas from financial modelling to artificial intelligence development, such as the Markov Chain Monte Carlo method [5].

This research uses the numerical approach of Monte Carlo methods to model outcomes to randomly played games of a generalised version of noughts and crosses. All simulations were conducted using the 'Numpy' and 'Scipy' package in Python.

II. Theory

A. Noughts and Crosses

Classically, in its simplest form, the game proceeds as follows: two players take turns to mark their tokens (either a nought or a cross) on an empty cell of a three by three grid. The game stops when there are three similar tokens along any row, column or diagonal, with the winning player the one who achieves this. This is illustrated in Fig.1.

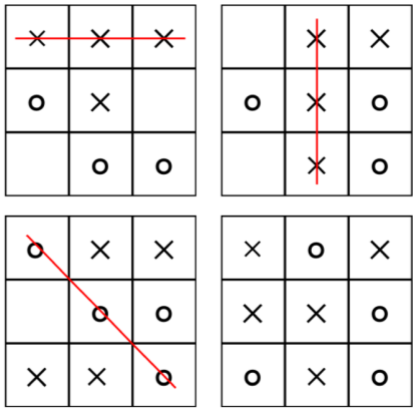

Figure 1: Illustration of typical 3×3 noughts and crosses games. The win condition includes any three identical tokens placed in a row, column or diagonal. A draw is reached when all spaces are filled without any player achieving the win condition. [11]

Due to the relatively small number of possible board states, the three by three game is trivially strongly solvable. Many variations of the game exist, most of which have increased complexity. Some of these variants have been solved rigorously [6], while others have solutions to specific cases [7]. In the variant used in this paper, the core rules are kept but generalised to $n \times n$ sized boards and the win condition being m tokens in row.

While the number of possible board states increases exponentially with increasing board size, it is plausible that certain variables – such as first player advantage – will exhibit trends as board size and win condition is varied.

B. Monte Carlo Simulations

The Monte Carlo simulations used to analyse the noughts and crosses games rely on generating random move sets and measuring the outcome for each [4]. These randomly generated simulations offer a way to model the outcomes of the game without delving into deep analysis of it. Use of the Mersenne Twister pseudorandom generator included in the 'Numpy' library to generate random move sets is able to yield accurate estimates for the mean [8].

III. METHOD

A. Simulation of Random Games

Monte Carlo simulations were used to investigate game length and advantage of starting in various positions in different game settings. Repeated randomly played games were simulated and the results of all games were analysed to draw conclusions on the investigated variables. The algorithms used to simulate and analyse the games made extensive use of Numpy arrays and libraries.

The simulation of randomly played games involved generating random moves for each game using the random number generator in the 'Numpy' library. Each move was played sequentially, and the state of the game was stored in a two dimensional $n \times n$ array. Checking for the win conditions proved to be the most resource intensive step due to board size increasing in quadrature and the need to check after every move. Several methods of simulation with different win checking algorithms were developed in an effort to optimise the process.

The initial win checking algorithm relied on checking the rows, columns and diagonals of every $m \times m$ sub-grid in the $n \times n$ game board for m tokens in a row, as illustrated in Fig. 2. The time required to execute this algorithm increased exponentially with increasing values of n and $(n - m)$ as the game length and number of sub-grids increased accordingly. This motivated the use of other algorithms to optimise the simulations.

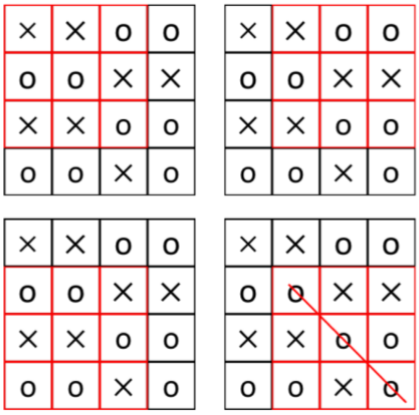

Figure 2: The initial win checking algorithm checks every $m \times m$ sub-grid in the $n \times n$ game board for m tokens in a row. In this illustration, $n = 4$ and $m = 3$. The number of sub grids to check is $(n - m + 1)^2$. [11]

Similarities between the identification of m identical tokens in a row and morphological binary image processing inspired the use of the image processing functions. Identification of a shape of objects is prevalent in image processing where pattern detection is crucial. This introduced the possibility of using sophisticated image processing techniques to do the heavy lifting.

Use of the binary erosion operation in the multidimensional image processing library of the 'Scipy' package proved to be significantly faster than the previous method at higher values of $(n - m)$. This trend is observed in Fig. 3, where the time taken per trial remained similar for different $(n - m)$ values when using the binary erosion method. For both methods, the time taken per trial increased with increasing n. However, the increase is at a slightly slower rate for the image processing tool, as observed from the gentler gradient of its trendline.

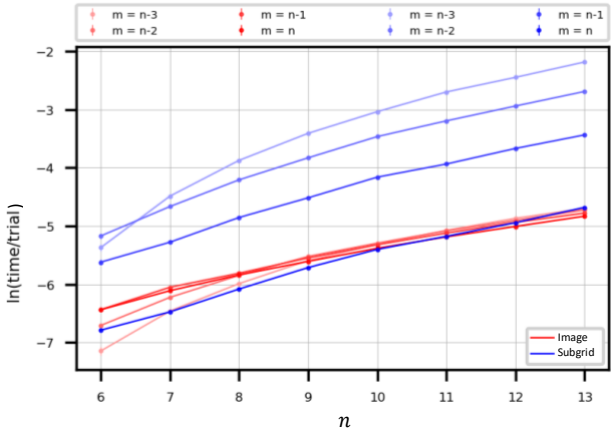

Figure 3: Plot of time taken per trial for increasing values of n , where time is on a logarithmic scale. Red represents the algorithm using the binary erosion method from the image processing library, and blue represents the initial method that checks every m × m subgrid. [11]

To further optimise the execution time, vectorisation techniques were incorporated into the algorithm. A technique commonly used to speed up algorithms involving repetitive calculations, vectorisation involves the simultaneous simulation of multiple games or trials. This was achieved by stacking multiple $n \times n$ two-dimensional game arrays to form a $x \times n \times n$ three-dimensional array, where x is the number of simultaneous trials.

Vectorisation of the calculations was able to create significant efficiency gains for the algorithms in all cases under the right conditions. As observed in Fig. 4, the optimal number of simultaneous trials is approximately $e^6 \approx 500$, and results in a time taken per trial that is of several orders of magnitude shorter relative to looping through trials individually.

When looping each trial individually, time taken per trial remains constant, which can be explained by the linear nature of looping identical processes. However, the time taken per trial exhibits a vastly different relationship when the algorithm is vectorised. Longer time taken for small numbers of trials executed can be attributed to the greater resources required to initialise the more complex arrays used in vectorisation. On the other end of the spectrum, the efficiency lost in executing a large number of trials is likely due to the hardware or memory limitations of the machine used for the execution of the program. Between these dichotomies, the time taken per trial drops to a much lower value, and represents an opportunity for significant efficiency gains.

Considerations of the above factors resulted in the final algorithm making use of the binary erosion method on three dimensional arrays containing 500 simultaneous trials. The number of trials simulated was

$$500 \times n^2 , \tag{1}$$

under the following conditions:

$$3 \leq n \leq 10 , \tag{2}$$

$$0 \leq n - m \leq 4 , \tag{3}$$

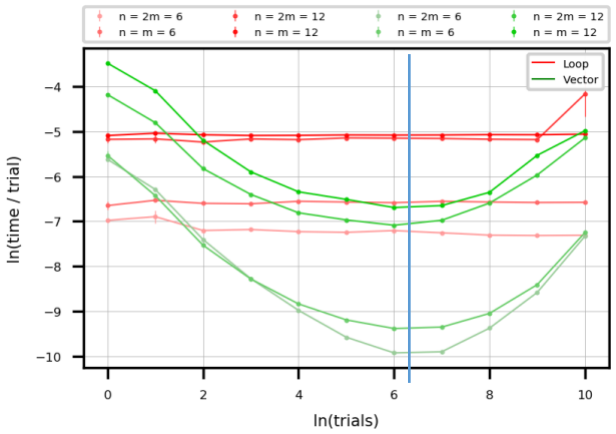

Figure 4: Plot of the time taken per trial as a function of number of trials run, plotted on a logarithmic scale. Red represents looping through each trial individually, while green represents the same algorithm but vectorised so all trials are run simultaneously. Different shades represent differing n and m values. The blue line indicates the approximate optimal number of trials. Errors are present but small. [11]

The results were analysed to investigate game length and advantage for different game board sizes and win conditions.

IV. Results, Errors and Discussion

A. Game Length

Game length, represented by the turn in which the game was won, increased non-linearly as board size increased. As shown in Fig. 5, the trend exhibited is possibly quadratic, which can be explained by the direct relationship between game length and board size, n^2.

The trend remains similar for various $(n - m)$ values, but with a consistent reduction for increased m values. A lower m value represents a win condition that can be fulfilled in a smaller number of moves, leading to shorter game lengths. The vertical translation indicates a linear relationship between m and game length.

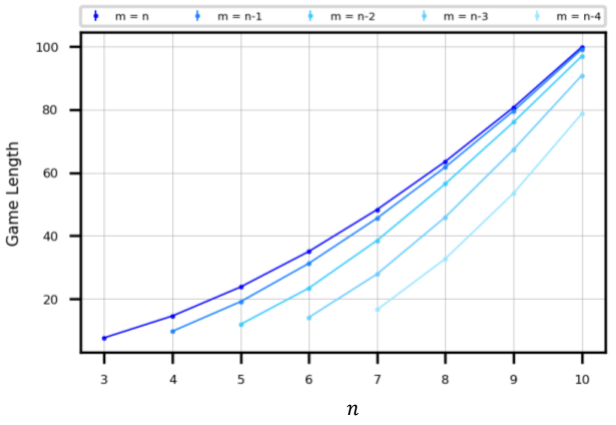

Figure 5: Game Length as a function of n, the length of the board. Similar trends are observed for decreasing values of m, but with a vertical downward displacement. Error bars are present but too small to be seen. [11]

B. Advantage

Advantage is a metric used to measure the difference in probability of winning between the two players. Since the starting player has a greater or similar number of tokens on the board at any point in time, starting first results in a higher rate of winning in most circumstances. The advantage calculated is that of the starting player and is determined, for a given board size and win condition, by:

$$Advantage = P_1 - P_2\,, \tag{4}$$

where P_1 and P_2 is the probability of the first and second players winning respectively. They are determined by taking the ratio:

$$P_i = \frac{No.\ of\ games\ won\ by\ Player\ i}{No.\ of\ games}\,, \tag{5}$$

where $i \in \{1,2\}$.

The advantage was calculated for every possible starting position for various values of m and n. A heatmap representing the advantage was produced, as shown in Fig. 6. The general trend for a $n \times n$ board where $m = n$ is that starting on the main diagonals results in a significantly greater advantage. For odd n, the main diagonals intersect at the middle square, and the middle square shows a particularly large advantage relative to the rest of the board. For even n, no intersections occur and the advantage remains similar along the main diagonals.

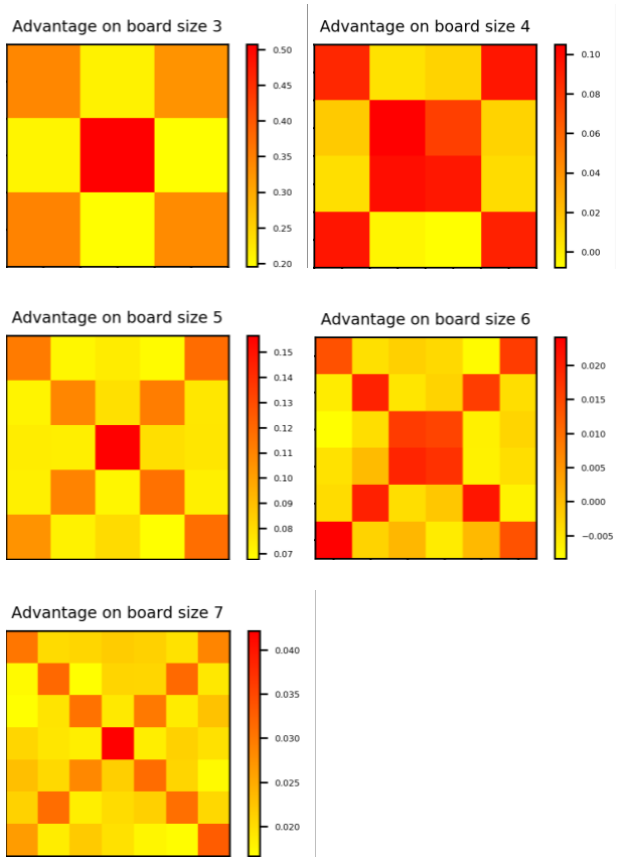

Figure 6: Heat map showing the advantage of starting in different positions on the board. Red means greater advantage, and yellow less. [11]

The heat map, like the board, is symmetrical about four axes: vertical, horizontal and two diagonals. This implies that the number of possible scenarios to consider can potentially be greatly reduced by recognising symmetrical board states and eliminating repeated ones.

The advantage heat map is significantly different when $m = n - 1$, as illustrated in Fig. 7. The corners no longer represent an advantageous position to start, and instead the positions adjacent to them become more attractive. The change in distribution can be explained by relating the advantage of a position to the number of rows, columns and diagonals of length m that overlap at that position. A common trend is the greater advantage for positions in closer proximity to the centre.

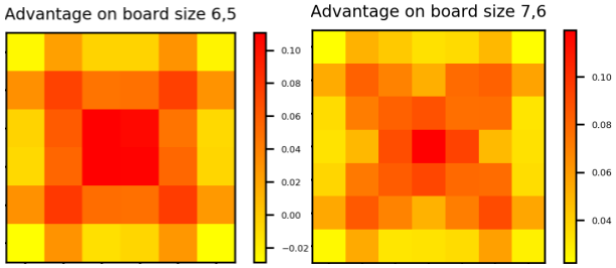

Figure 7: Heat map when $m = n - 1$ for values of $n = 6$ and $n = 7$. There is significant change relative to the case where $m = n$ as shown in Fig. 6. [11]

By considering the mean of the advantages of each starting position, an expectation value can be derived for the advantage of starting first. It is intuitive that playing first would entail a benefit since the first player always has more or equal number of tokens on the board. However, this advantage decreases with increasing n due to the token advantage making up a smaller proportion of tokens on the board. As $n \to \infty$, it is only reasonable that the advantage tends to zero since the probability of any player even winning tends to zero as well. The expected trend is observed in Fig. 8.

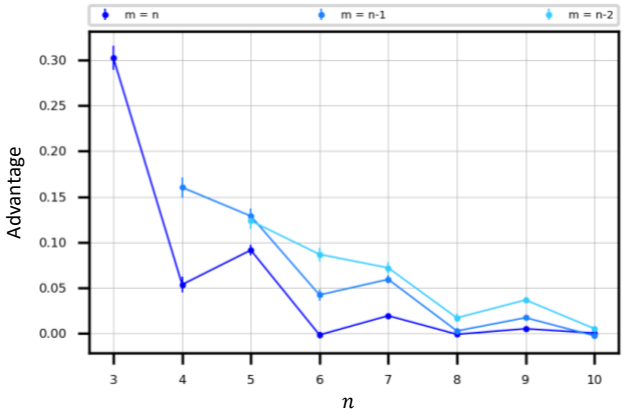

Figure 8: The advantage of starting first as a function of n for cases where $n \geq m \geq n - 2$. [11]

An exponentially decaying trend is exhibited, but the advantage for even n is markedly lower than on adjacent odd n. This inconsistency could be caused by the differences in board symmetry for different n. Unlike boards with even n, those with odd n are symmetrical about the central row, column or diagonal.

C. *Errors*

Errors for the advantage stem from the random nature of Monte Carlo methods. Simulating trials from a random number generator leads to a sample distribution from which the population mean and standard error can be estimated [9]. The population standard error σ_0 is determined from the sample error σ from the relation:

$$\sigma_0 = \frac{\sigma}{\sqrt{N}}, \tag{5}$$

where N is the number of trials conducted to generate the data sample. As such, a large number of trials had to be conducted to ensure that errors were small and that the sample mean was close to that of the population. A factor of 500 was chosen in (1) so as to achieve a small σ_0. This resulted in a trade-off between the accuracy of the results and the time taken for the execution of the simulations.

V. Conclusion

Analysis on generalised noughts and crosses games was conducted using Monte Carlo simulations. As with the execution of all Monte Carlo methods, a balance has to be struck between conducting sufficiently many calculations for an accurate result and keeping calculation time reasonable. It was concluded that game length followed the expected trend for various game conditions. The advantage of starting in various positions for different board states exhibited general patterns that gives insight into optimal strategies. The analysis, however, is severely limited in its application to real world scenarios due to the lack of a strategy in making moves. Further work with considerations of strategic plays can yield better insights on the optimal play, paving the way to the development of an artificial intelligence capable of making strategic moves.

VI. Bibliography

[1] J. v. Neumann and O. Morgenstern, Theory of games and economic behaviour, Princeton: Princeton University Press, 1947.

[2] C. Zaslavsky, Tic Tac Toe: And Other Three-In-A Row Games from Ancient Egypt to the Modern Computer, Harpercollins, 1982.

[3] M. Nagy and N. Nagy, "Quantum Tic-Tac-Toe: A Genuine Probabilistic Approach," *Applied Mathematics*, vol. 3, pp. 1779-1786, 2012.

[4] D. P. Kroese, "Monte Carlo methods," *WIREs Computational Statistics*, vol. 4, no. 1, pp. 48-58, 14 December 2011.

[5] D. v. Ravenzwaaij, P. Cassey and S. D. Brown, "A simple introduction to Markov CHain Monte-Carlo sampling," *Psychonomic Bulletin & Review*, vol. 25, no. 1, pp. 143-154, February 2018.

[6] J. Beck, Combinatorial Games: Tic-Tac-Toe Theory, New York: Cambridge University Press, 2008.

[7] J. Cardinal, S. Collette, H. Ito, M. Korman, S. Langerman, H. Sakaidani and T. Perouz, "Cannibal Animal Games: a new variant of Tic-Tac-Toe,"

Journal of Information Processing, vol. 23, no. 3, pp. 265-271, 2015.

[8] J. C. Hill, "Bias in Monte Carlo Simulations Due To Pseudo-Random Number Generator Initial Seed Selection," *Journal of Modern Applied Statistical Methods*, vol. 10, no. 1, pp. 29-50, 2011.

[9] A. E. Feiguin, "Monte Carlo Error Analysis," in *Phys 5870: Modern Computational Methods in Solids*, Boston, Northeastern University, 2009, p. Section 10.

[10] B. L. D. Physics, First Year Laboratory Manual, London: Imperial College London, 2017.

[11] S. C. Tan, *Monte Carlo Noughts and Crosses*, 2018.

VII. Acknowledgements

Assistance provided by partner M Lin and mentor D Ho is greatly appreciated.