Tic-Tac-Toe To The Third Dimension

Project 8-18

Hwa Chong Institution

Chew Yan Kee 4A2 (05)

Lim Jin Sian 4i3 (21)

Tan Chun Ern 4A2 (22)

Contents

1 Introduction

1.1 - Introduction and Rationale

3D tic-tac-toe is an abstract strategy board game, generally for two players. It is similar in concept to traditional tic-tac-toe but is played in a cubical array of spaces. Players take turns placing their markers, usually crosses or naughts, in blank spaces in the array. The first player to achieve a complete row of their own markers in a row wins. The winning row can be horizontal, vertical, or diagonal on a single board as in regular tic-tac-toe, or vertically in a column, or a diagonal line through the boards. In the project, we aim to analyse solutions for multiple variations of 3D tic-tac-toe and hope that our research is able to generate interest in the game itself.

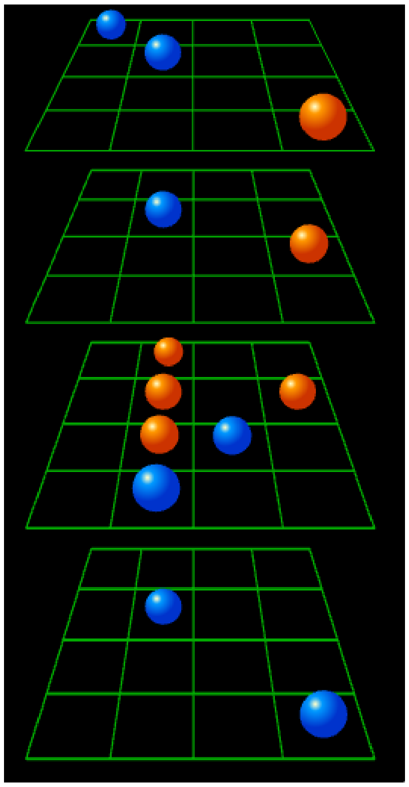


Fig.1: An example of a win by blue in 4x4x4 tic-tac-toe

1.2 - Terminology

Winning line (L)	The winning combinations of a board
Strongest Point Line (SPL)	The winning lines that cross through such the point that is contained in the most winning lines
NxNxN Tic-Tac-Toe	Three-dimensional Tic-Tac-Toe with N rows, columns and height.
Forced sequence	A sequence where the first player or the second player is able to force a win.
k^n positional game	A game seen as an n-dimensional hypercube with k spaces on an edge
‘Fair game”	A game where the second player is able to force a draw

1.3 - Field of Mathematics involved

The main field of mathematics involved in the project would be Game Theory.

2 Objectives

1: To explore on the winning strategies of the classic 3x3x3 Tic-Tac-Toe

2: To manipulate the rules such that it will be a fair game for both players

3: To extend our research findings into 4x4x4 Tic-Tac-Toe

2.1 - Research Questions

1. What is the optimal game theory play for 3x3x3 Tic-Tac-Toe?

2. How do we decide on the way to manipulate the rules, and how will the rules affect the gameplay?

3. How do we extend our game optimal theory play for 3x3x3 Tic-Tac-Toe to 4x4x4 Tic-Tac-Toe?

3 Literature Review

3.1 - Strategy stealing argument

From game theory (Blackwell and Girshick, 1954), we know that for k^n positional games, the second player can't force a win, as the following argument shows. Suppose that the second player (P2) is using a strategy S which guarantees a win. The first player (P1) places an X in an arbitrary position. P2 responds by placing an O according to S. But if P1 ignores the first random X, P1 is now in the same situation as P2 on P2's first move: a single enemy piece on the board. P1 may therefore make a move according to S – that is, unless S calls for another X to be placed where the ignored X is already placed. But in this case, P1 may simply place an X in some other random position on the board, the net effect of which will be that one X is in the position demanded by S, while another is in a random position, and becomes the new ignored piece, leaving the situation as before.

Continuing in this way, S is, by hypothesis, guaranteed to produce a winning position (with an additional ignored X of no consequence). But then P2 has lost – contradicting the supposition that P2 had a guaranteed winning strategy. Such a winning strategy for P2, therefore, does not exist, and tic-tac-toe is either a forced win for P1 or a tie.

3.2 - Computer-assisted proof of 4x4x4 Tic-Tac-Toe

In 1980, Patashnik performed an analysis on the winning strategies of 4x4x4 Tic-tac-toe by utilizing a computer program, which resulted in a strategy including move choices for 2929 difficult "strategic" positions, plus assurances that all other positions that could arise could be easily won with a sequence entirely made up of forcing moves (Patashnik, 1980). From his findings, we found out that we can check whether k^n games belong to these three classes. Class 1 consists of those games for which draws are impossible even under nonoptimal play; that is, no draw position exists. This means that the first player can always force a win in these games, since, as we saw earlier, the second player can't due to the strategy stealing argument. Class 2 consists of those games for which a draw position exists but for which the first player can nevertheless force a win. Class 3 consists of those games for which the second player can force a draw. Thus, under optimal play, games in classes 1 and 2 are first-player wins, and games in class 3 are draws. It has also introduced the concept of winning lines and strongest-point lines, which will be denoted as L and SPL respectively. By calculating the amount of L and SPL there is in a game, we can check if the game belongs to these three classes by checking whether $SPL + L < 2^k$ or if $k \geq 2 \times SPL$. Besides that, Patashnik has also developed his own computer algorithm to search for forced sequences, or sequences where is a sequence of moves from a given position, in which Player O must continually block Player X's three-in-a-row until at some move he or she must simultaneously block two such threes-in-a-row, as this is impossible, Player X wins. These force sequences are then inputted into an algorithm to search for distinct-position trees based on that algorithm, which is the complete game tree of the game after eliminating the redundant positions in the game tree, which prevents the algorithm from blowing up.

4 Research Findings

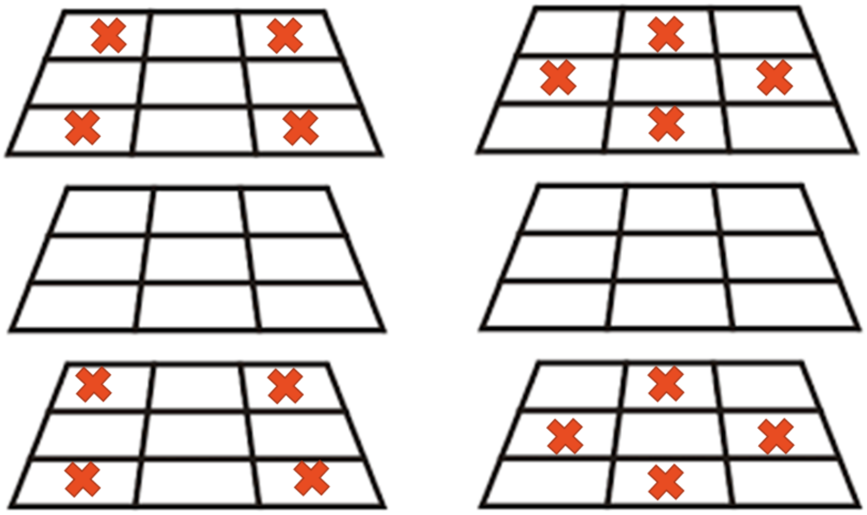
4.1 Objective 1

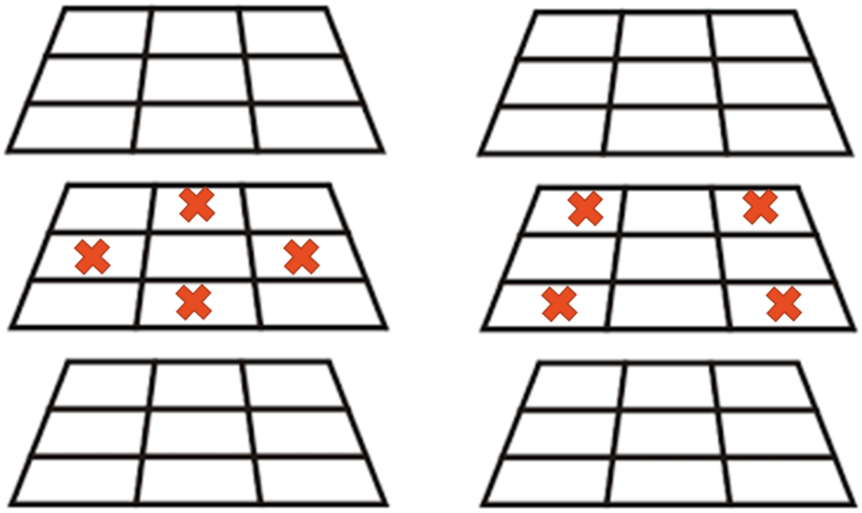
4.1.1 Automorphisms of 3x3x3

In order to solve 3x3x3 Tic-Tac-Toe, our group decided to narrow down the amount of cases that we have to consider. To do that we have to consider automorphisms of the board of 3x3x3 Tic-Tac-Toe. An automorphism of the board is a symmetry of the object, and a way of mapping the object to itself while preserving all of its structure. This means that an automorphism of the board would eventually map out the same structure of the original by rotating the board.

Looking at it on another way, a player has a strategy for playing from position

P, the transformation F gives him a recipe for transforming that strategy into one for playing from position PF. The essential property that F possesses is that it preserves 3 in a row: Three distinct points a, b, c of the board are collinear if and only if aF, bF, cF are collinear. An automorphism of 3x3x3 Tic-Tac-Toe is a mapping of the board onto itself that preserves 3 in a row. Below, our group has presented the groups of positions that are automorphisms of each other:





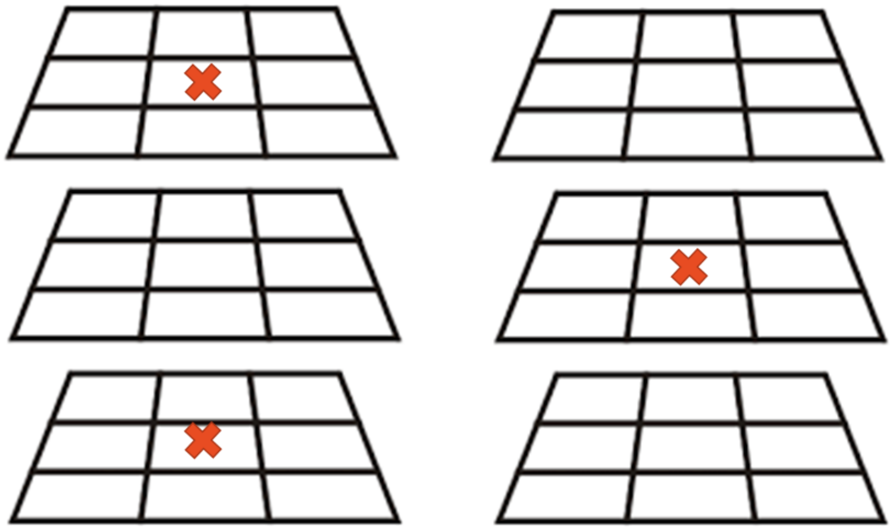
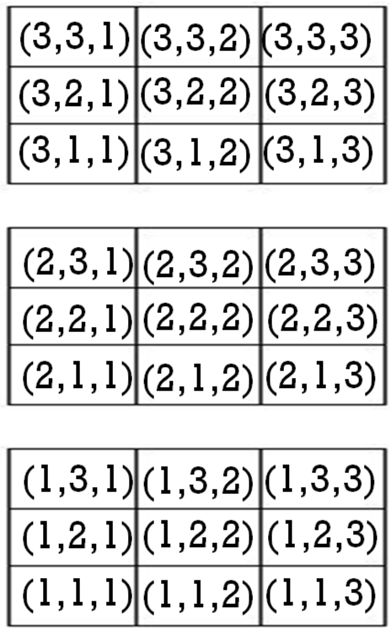


Fig.2: The automorphisms of the 3x3x3 board, the marks presented on the same board are equivalent to each other.

4.1.1 Solving 3x3x3

Now that we have eliminated all equivalent positions for 3x3x3 Tic-Tac-Toe, we can develop a strategy for the first-player such that it can force a win by creating two 2-in-a-row lines in a turn regardless of the positions of the second player's marks by using a brute-force strategy due to the small number of cases that needs to be considered. After narrowing down the winning cases by eliminating all automorphisms, this leaves us with only four winning cases.



(3,3,1)	(3,3,2)	(3,3,3)
(3,2,1)	(3,2,2)	(3,2,3)
(3,1,1)	(3,1,2)	(3,1,3)

(2,3,1)	(2,3,2)	(2,3,3)
(2,2,1)	(2,2,2)	(2,2,3)
(2,1,1)	(2,1,2)	(2,1,3)

(1,3,1)	(1,3,2)	(1,3,3)
(1,2,1)	(1,2,2)	(1,2,3)
(1,1,1)	(1,1,2)	(1,1,3)

Fig.3: Coordinates of the spaces in a 3x3x3 board.

Let F be the first player's move and S to be the second player's move. An example of a sequence of moves would be similar to this:

F(1,1,1) S(1,1,2)

Corresponds to: First player places a mark at point (1,1,1), followed by the second player who places a mark at point(1,2,2).

Here, I will present the four winning cases of 3x3x3 Tic-Tac-Toe.

Case 1:

F(2,2,2) S(1,2,2) F(1,1,1) S(3,2,2) F(1,1,3), First player wins as there are two winning lines

Case 2:

F(2,2,2) S(1,2,1) F(1,3,1) S(3,2,3) F(1,3,3), First player wins as there are two winning lines

Case 3:

F(2,2,2) S(2,2,1) F(2,3,1) S(2,1,3) F(2,3,3), First player wins as there are two winning lines.

Case 4:

F(2,2,2) S(2,3,1) F(3,2,2) S(1,2,2) F(3,3,2),First player wins as there are two winning lines

Thus, 3x3x3 Tic-Tac-Toe is always won by the first player.

4.2 Objective 2

4.2.1 Manipulation of Rules

For this objective, our group has decided to attempt to manipulate the rules of 3x3x3 Tic-Tac-Toe in order to make the rule a fair game or allow the second player a way to force a draw. A fair game means that both players have an equal chance of winning, instead of one player being able to force a win. To do this, our group resorts to restricting certain spaces or points on the 3x3x3 Tic-Tac-Toe board such that the first player is unable to force a win. As there are 27 spaces in the board, there are altogether 2^{27} - 2 cases of restricted 3x3x3 tic-tac-toe, without including automorphisms. The two cases refer to the cases where all spaces are restricted and all spaces are unrestricted.

4.2.2 Filtration of Cases

Since there are a lot of possible cases, we are going to use a computer-aided approach to help us find the variations out of $2^{27} - 2$ cases that are fair. Due to the sheer number of cases involved, we would attempt to narrow down the number of cases before doing a brute force search. We referred to Patashnik's computer algorithm, and modified his algorithm such that it suits our needs.

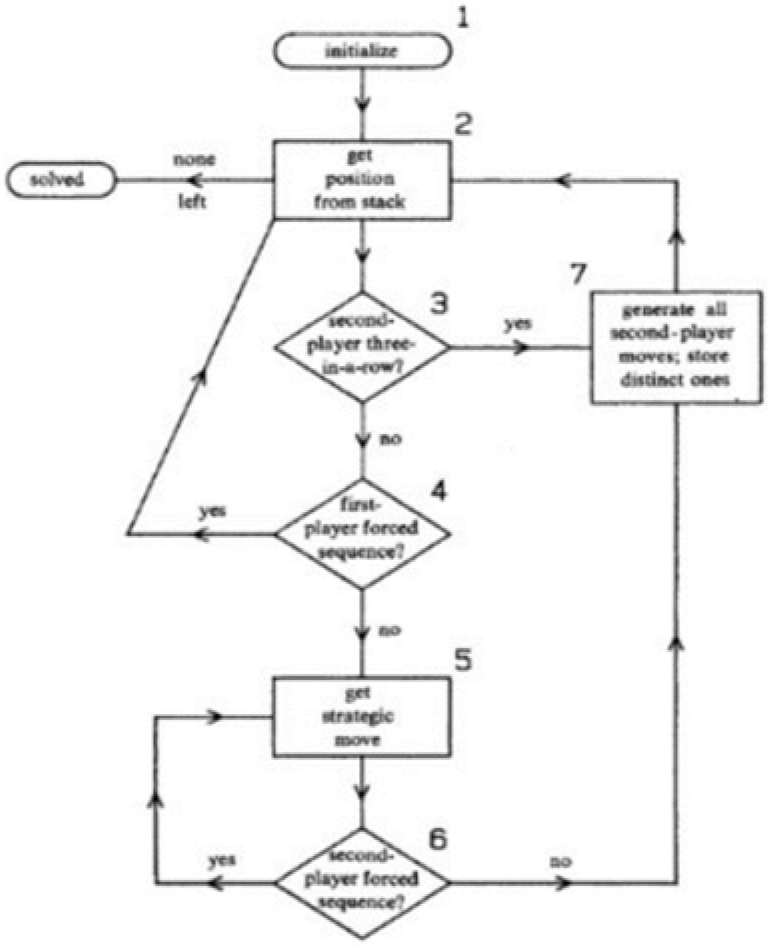

1
initialize
2
get position from stack
none left
solved
3
second-player three-in-a-row?
yes
7
generate all second-player moves; store distinct ones
no
4
first-player forced sequence?
yes
no
5
get strategic move
6
second-player forced sequence?
yes
no

Fig.4: Patashnik's flowchart of his algorithm

According to Patashnik's article, in a k^n tic-tac-toe game, the second player can force a draw if k> 2 x SPL or 2^n > SPL+L, therefore, we can filter out the variations that fulfils either of this condition using the algorithm, which are the variations that the second player can force a draw, which we considered to be a "fair game". If it doesn't fulfill either of the conditions, then we have to brute-force to see if the variation is one that the first player can force a win or one that the second player can force a draw. Using the two equations above allows us to increase the efficiency of our algorithm as fewer cases would be required to go through the brute force process, which requires a significant amount of computing power and time.

4.2.3 Calculation of L and SPL of 3 x 3 x 3 Tic-Tac-Toe

Let k^n be the dimension of the Tic-Tac-Toe, which in this case will be a hypercube H. we have discovered that the Winning Lines (L) can be calculated by applying the formula: $\{ (k+2)^n - k^n \} / 2$. Embed H in a $(k+2)^n$ hypercube H' so that H' has a layer of spaces on each side of H and let S be the shell of points in H' that are not in H, so there are $(k+2)^n - k^n$ points in S.

Knowing that k^n represents the dimension of the cube, the dimension of the cubic array on the diagram's left side will assume the dimension of K^n. The dimension of the cubic array on the right side of the diagram will therefore assume a cubic array of $(k+2)^n$.

Consider a k^n hypercube H where $k>1$. For each winning line l in H, its unique extension l' to H' contains exactly two points of S, one at either end of l'. Furthermore, for any point in S, exactly one winning line l of H extends to it. Hence, the number of winning lines of H is equal to half the points of S, or $\{(k+2)^n - k^n\} / 2$, as claimed.

When $K > 2(SPL)$ or $L + SPL < 2^n$, the second player is able to force a draw for the Tic-Tac-Toe game.

In order to confirm that a case satisfies the above conditions, we first allocate the spaces with coordinates, as in Fig.3. Then, we add all the combinations of spaces that form a winning line into a set W. When we are calculating L of a case, we check for combinations of unobstructed spaces that are elements in set W. Those that are elements in set W would be considered a L. For the SPL, we would check for the coordinate that appeared the most number of times in the combinations of coordinates that are an element in set W, and calculate the number of combinations that it has appeared in.

After finding the L and SPL of a case, we can calculate whether a case fulfills the given conditions, and thus whether the case allows for the second player to force a draw. However, even if a case does not fulfil the two conditions, we are still not entirely sure whether the case allows a forced draw for the second player.

4.2.4 Finished algorithm

Our finished algorithm would be separated into two parts, the first part would be to calculate the L and the SPL of a variation of the board in order to determine whether the variation results in a fair game, while the second part of the algorithm would filter out cases that do not fulfil the two conditions but would still be able to allow a forced draw by the second player.

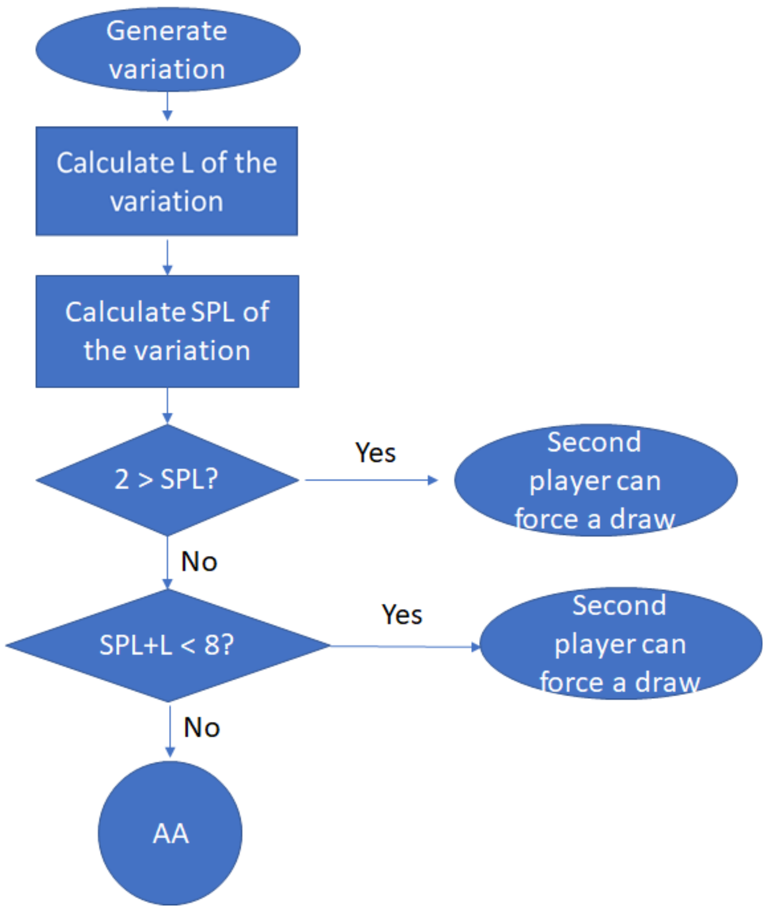

Generate variation
Calculate L of the variation
Calculate SPL of the variation
2 > SPL?
Yes
Second player can force a draw
No
SPL+L < 8?
Yes
Second player can force a draw
No
AA

Fig.5.1: First part of the algorithm

The first part of the algorithm generates a variant of the 3x3x3 board with some spaces obstructed, it then calculates the L and SPL of the variant. If the variation fits the conditions, it terminates the algorithm and generates another variation, or else it feeds the variation into the second part of the algorithm if it does not fit the conditions.

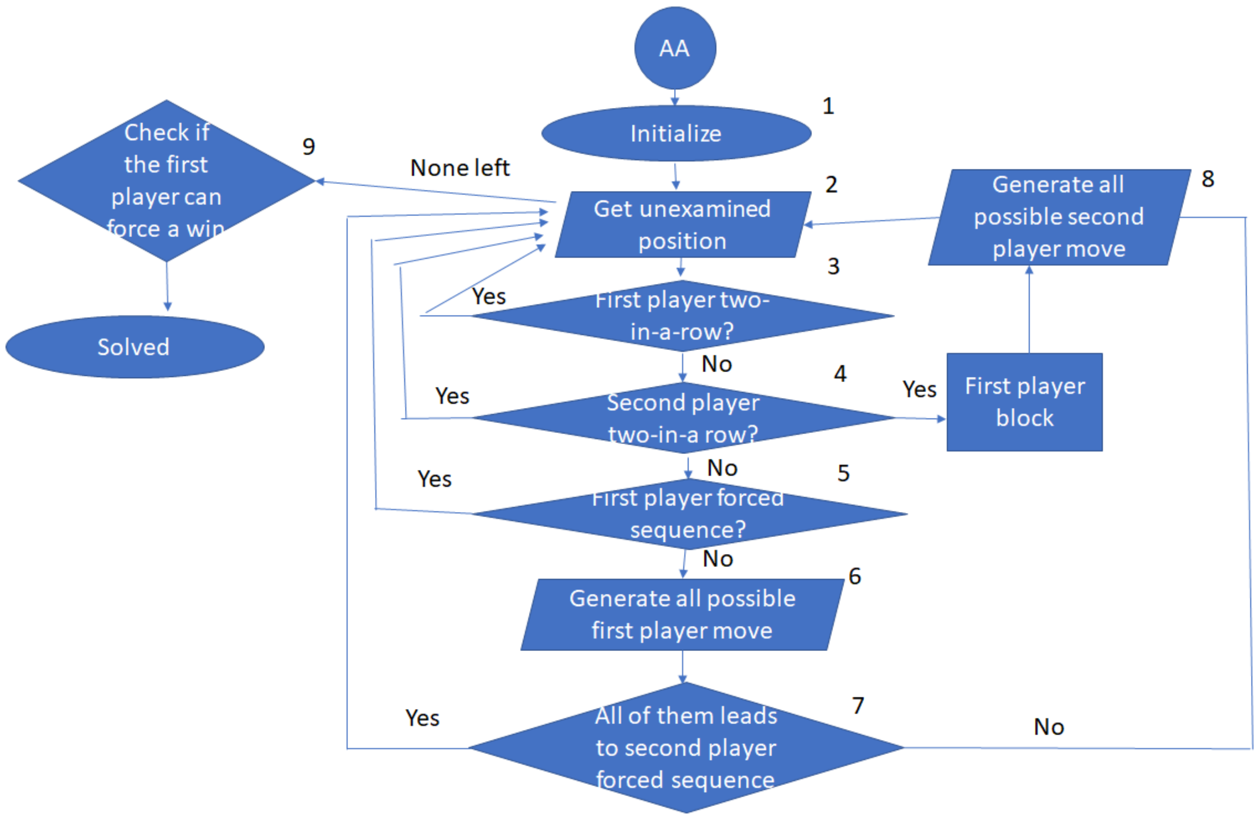

AA
Initialize
1
Get unexamined position
2
None left
Check if the first player can force a win
9
Solved
First player two-in-a-row?
3
Yes
No
Second player two-in-a row?
4
Yes
Yes
First player block
Generate all possible second player move
8
No
First player forced sequence?
5
Yes
No
Generate all possible first player move
6
All of them leads to second player forced sequence
7
Yes
No

Fig.5.2: Second part of the algorithm

At the second part of the algorithm, it would go through the following steps.

Step 1: We store the starting position of the variation into a set A and start running the algorithm.

Step 2: We will get an unexamined position from set A and go to step 3. If there are no unexamined positions left, go to step 9.

Step 3: Check for a first player two-in-a-row. If one exists, the first player will take the space that is unoccupied in the row of the two-in-a-row and the position will be stored into set A as examined positions and go to step 2; otherwise go to step 4.

Step 4: Check for a second-player two-in-a-row (step 7 ensures at most one). If one exists, block the two-in-a-row by occupying the last square and go to step 8; otherwise go to step 5.

Step 5: Check for a first-player forced sequence. If one exists, the first player occupies the space that will cause the force sequence to happen and stores its positions as examined positions to set A and go back to step 2; otherwise go to step 6.

Step 6: Generate all possible first player moves; then go to step 7.

Step 7: Check if all of the possible first player moves will lead to a second-player forced sequence. If yes, store all the positions as examined positions in set A and go back to step 2; otherwise go to step 8.

Step 8: Generate all possible second player moves. Put the resulting positions into set A as unexamined positions and go back to step 2.

Step 9: Check if the first player can force a win.

4.2.5 Checking for first player win

At step 9, all of the positions in set A are examined.

The last move of any of the positions are either made by the first player or second player. Let's say there are r number of positions in set A. Set all these positions as p_1, p_2, p_3, … , p_{r-1}, p_r and define i-position as a position where there are i number of spaces occupied by either the first player or the second player. As an example, if p_1 is a 20-position, that means in p_1, there are 20 spaces occupied by players and the first player and second player occupies 10 spaces each.

The starting position is a 0-position.

Note that every position in set A is reached with a distinct order of move.

Set every p_t ($0 < t < r+1$, and t is a positive integer) as a q_t-position. When the (q_t-2)th move was made in accordance to the move order to reach the position p_t, we define the position as the (q_t-2)-position of p_t.

Notice that we already know if the first player can forced a win for all of the positions in set A (p_1, p_2, p_3, … , p_{r-1}, p_r) All these positions are either an even number-position that the first player cannot force a win, an odd number-position that the first player cannot force a win or an odd number-position that the first player can force a win.

If the position p_u is an even number-position that the first player cannot force a win, we know that the first player cannot force a win when the position (q_{u-}1)-position of p_u is reached. Therefore, we remove all even number-position p_u from set A and replace it with the (q_u-1)-position of p_u.

Now, all the positions in set A are either an odd number-position that the first player cannot force a win or an odd number-position that the first player can force a win.

After that, the algorithm will select a position p_s from set A where p_s is one of the positions with the most spaces occupied. If the position p_s is an odd number-position that the first player can force a win, check through searching from set A if the first player can force a win for all possible second player moves after the (q_s-2)-position of p_s. If yes, it means that the first player can force a win when the position (q_s-2)-position of p_s is reached and therefore we remove all position that is one of the possible outcome after the (q_s-2)-position of p_s from set A and replace it with the (q_s-2)-position of p_s. If no, it means that the first player cannot force a win when the position (q_s-2)-position of p_s is reached and therefore we remove all positions that are one of the possible outcome after the (q_s-2)-position of p_s from set A and replace it with the (q_s-2)-position of p_s.

If the position p_s is an odd number-position that the first player cannot force a win, check through searching from set A if the first player can force a win after the (q_s-1)-position of p_s. If not, it means that the first player cannot force a win when the position (q_s-2)-position is reached. Therefore we remove all positions in set A that are one of the possible outcomes after the (q_s-1)-position of p_s replace it with the (q_s-2)-position of p_s. If yes, it means that the first player can force a win when the position (q_s-1)-position of p_s is reached and therefore we remove all possible outcome after the (q_s-1)-position of p_s from set A except one of the position where the first player can force a win.

Repeat the process until there is a 1-position produced that the first player can force a win, this means that the first player can force a win.

If there is no 1-position that the first player can force a win, this means that the second player can force a draw.

4.3 Objective 3

4.3.1 Algorithm for restricted 4x4x4 Tic-Tac-Toe

Objective 3 would be similar to objective in that the same algorithm can be used to determine whether any variation of the 4x4x4 board would be able to allow a forced draw by the second player. However, due to the increased number of spaces, some parameters must be tweaked in order for the algorithm to accommodate the 4x4x4 board.

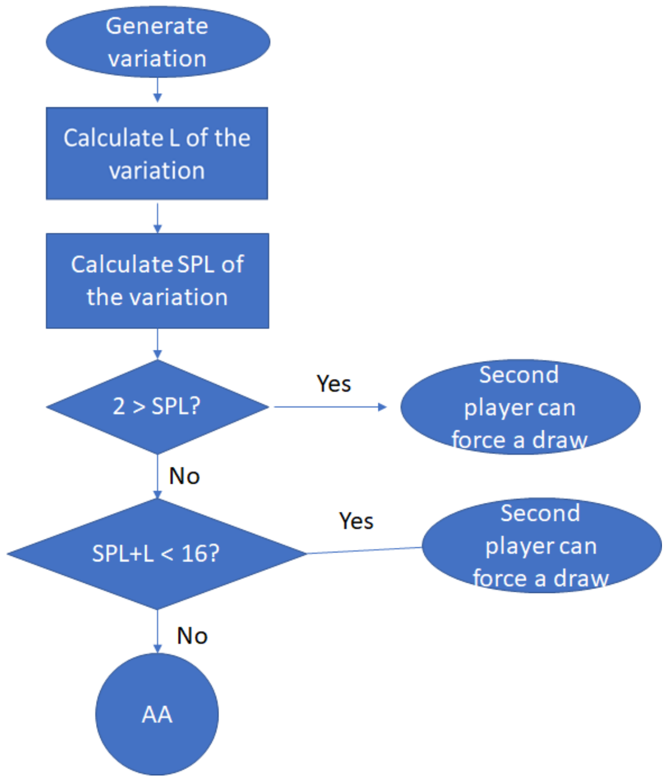

Generate variation
Calculate L of the variation
Calculate SPL of the variation
2 > SPL?
Yes
Second player can force a draw
No
SPL+L < 16?
Yes
Second player can force a draw
No
AA

Fig 6.1: Modified version of the first part of the algorithm

As the value of k in the 4x4x4 Tic-Tac-Toe board is 4 instead of 3, we would need to modify the second condition that SPL +L < 2^k. $2^k = 2^4 = 16$, thus for the algorithm accommodating 4x4x4 Tic-Tac-Toe, we must modify the second condition such that the algorithm instead checks whether the SPL + L of the board is less than 16.

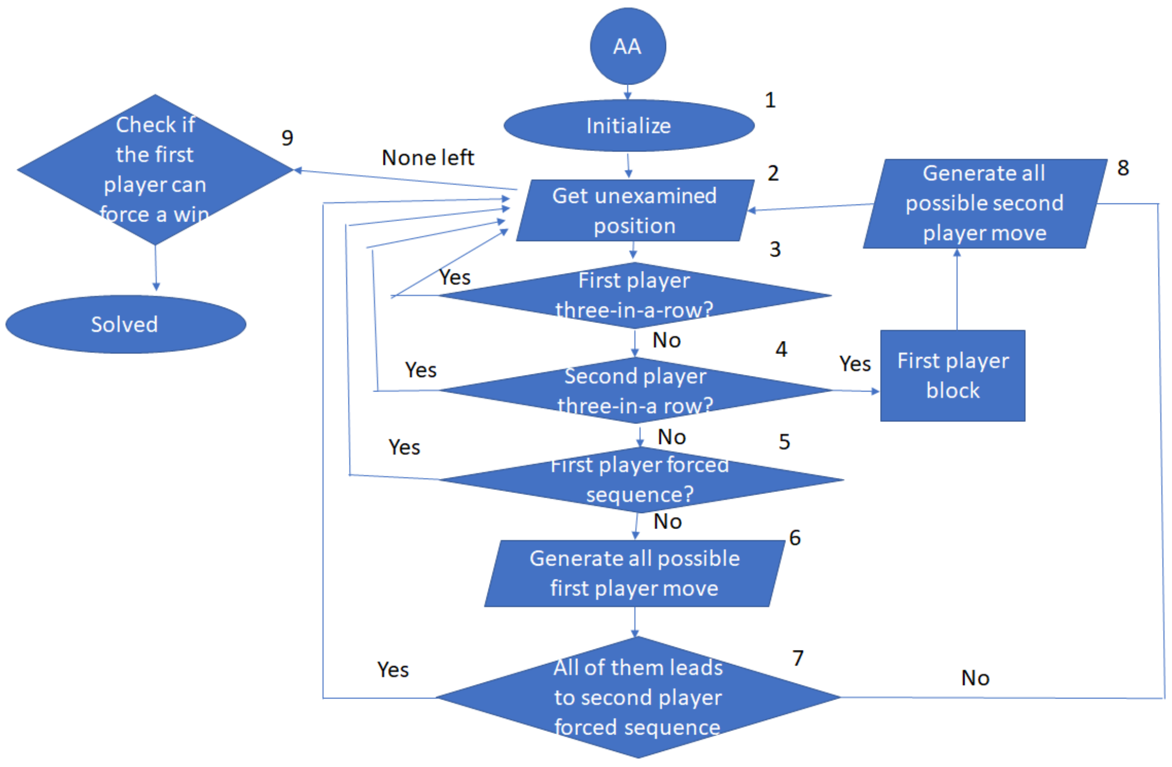

AA
Initialize
1
Get unexamined position
2
None left
Check if the first player can force a win
9
Solved
First player three-in-a-row?
3
Yes
No
Second player three-in-a row?
4
Yes
Yes
First player block
Generate all possible second player move
8
No
First player forced sequence?
5
Yes
No
Generate all possible first player move
6
All of them leads to second player forced sequence
7
Yes
No

Fig 6.2: Modified version of the second part of the algorithm

For the second part of the algorithm, the algorithm instead checks for three-in-a-rows present on the board instead of two-in-a-rows due to how 4x4x4 Tic-Tac-Toe is played.

4.3.2 Algorithm for restricted NxNxN Tic-Tac-Toe

During our research, we found out that we can further extend our algorithm such that it is able to generate variations for any NxNxN board and calculate whether it is able to allow a second player forced draw.

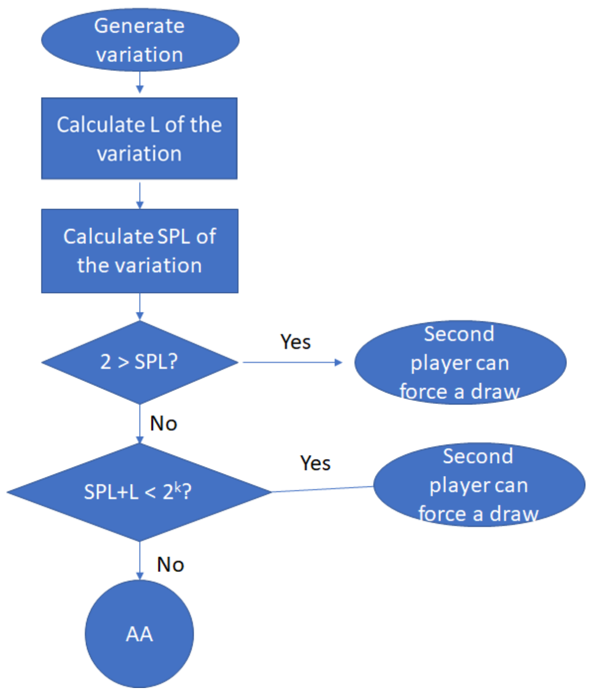

Generate variation
Calculate L of the variation
Calculate SPL of the variation
2 > SPL?
Yes
Second player can force a draw
No
SPL+L < 2^k?
Yes
Second player can force a draw
No
AA

Fig 7.1: Final version of first part of the algorithm

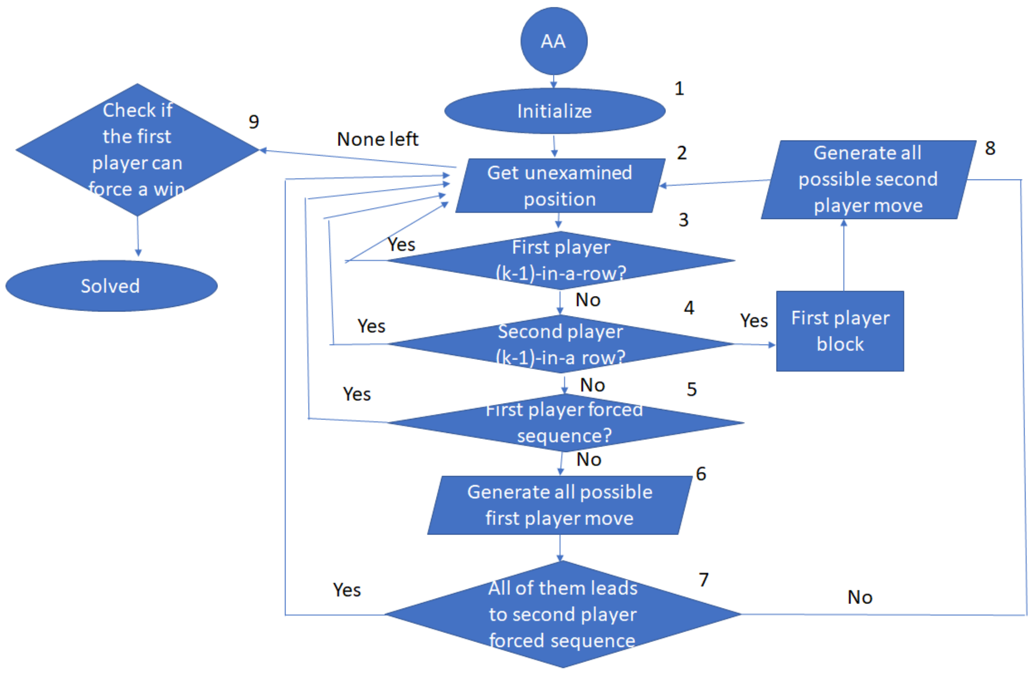

AA
1
Initialize
9
Check if the first player can force a win
None left
2
Get unexamined position
Generate all possible second player move
8
3
Yes
First player (k-1)-in-a-row?
No
4
Yes
Second player (k-1)-in-a row?
Yes
First player block
No
5
Yes
First player forced sequence?
No
6
Generate all possible first player move
Solved
7
Yes
All of them leads to second player forced sequence
No

Fig 7.2: Final version of second part of the algorithm

By adding a function in the first part of the algorithm that calculates the value of 2^k in the algorithm and modifying step 4 and step 5 such that it calculates the value of (k-1) ad uses that value to check for the number of pieces in a row that is equal to the value of (k-1), the algorithm is now able to prove whether a variation of the NxNxN board is fair for both players. However, as the value of n increases, the number of cases increases exponentially. The number of variations of a NxNxN board can be calculated using the function $2^c - 2$, where c is the number of spaces that are available on the original board, which is also the value of n^3 . Thus, the number of cases would increase exponentially as n increases in value.

Value of n	Number of cases
1	0
2	254
3	$2^{27} - 2$
4	$2^{64} - 2$
5	$2^{125} - 2$
6	$2^{216} - 2$
7	$2^{343} - 2$
8	$2^{512} - 2$
9	$2^{729} - 2$
10	$2^{1000} - 2$

Table 1: Table of number of cases against the value of n

As such, the efficiency of this algorithm would rapidly decline due to the amount of cases it has to calculate, and as such would not be suitable for calculating variations of NxNxN boards with a high value of n.

5 Conclusion

5.1 Summary and Conclusions

1. 3x3x3 Tic-Tac-Toe is proved to be a first player win and we managed to deduce all possible winning cases.
2. We managed to produce an algorithm that deduces whether a variation of a NxNxN board allows a second player forced draw, or essentially allows a “fair game”.

5.2 Limitations and Possible Extensions

Although we are able to come up with the algorithm that calculates whether a variation of the NxNxN board allows a “fair game”, we are unable to translate the algorithm into code due to our lack of coding expertise. Besides that, our group also lacks the computing power necessary to process such a large number of cases, and thus our group is unable to strongly solve restricted Tic-Tac-Toe. Possible extensions include using the algorithm as a framework to produce a program that allows users to freely generate variants of the

NxNxN board and calculates whether the generated variant is able to allow a second player forced draw.

6 References

Patashnik, O. (1980). Qubic: 4 × 4 × 4 Tic-Tac-Toe. Mathematics Magazine, 53(4), 202-216. doi:10.2307/2689613

D. Blackwell and M. A. Girshick, Theory of Games and Statistical Decisions, Wiley, New York, 1954, p.21