EXPLORING TIC-TAC-TOE VARIANTS

By

Alec Levine

A SENIOR RESEARCH PAPER PRESENTED TO THE DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE OF STETSON UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE

STETSON UNIVERSITY

2018

ACKNOWLEDGMENTS

This senior research project would not have been possible without the guidance of Dr. Friedman. I would also like to thank each of the Math professors I have had during my time at Stetson University: Dr. Vogel, Dr. Edwards, Dr. Friedman, Dr. Coulter, and Dr. Miles. Without your classes, my understanding of both game theory and math in general would not be as strong as it is today. Lastly, thank you to all my friends of family who played hundreds of games of Tic-Tac-Toe with me.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

EXPLORING TIC-TAC-TOE VARIANTS

By

Alec Levine

May 2018

Advisors: Dr. Friedman
Department: Mathematics and Computer Science

Tic-Tac-Toe is an ancient game that has been solved so many times that the optimal moves are common knowledge, and the game always ends in a draw. However, when the rules to the game are changed, even slightly, the winning strategy becomes completely different.

This senior research is focused on variations of Tic-Tac-Toe and the effects they have on the winning strategies. The first variations we solved were Tic-Tac-Toe boards of varying sizes. This included the original 3x3 game board, the original game board with additional squares added in different places, and every square board larger than the size of 3x3. The other Variations that we look into are 3x3x3 Tic-Tac-Toe; Otrio, where players can choose between three different sizes for their markers; Veto Tic-Tac-Toe, where each player can "veto" one move each turn; the Misère version of Tic-Tac-Toe, where the last player to make a move loses; and Notakto, where each player plays only X's and the last player to make a move loses.

CHAPTER 1: An Introduction to Game Theory

1.1. P-Positions and N-Positions

This senior research is on Tic-Tac-Toe variations and how to find the winning strategies for each of them. Before we can look at actual games, though, we must first look at some important aspects of game theory. Game theory is a branch of mathematics that has existed for many years and has been used to solve the winning strategies for many different games, such as Tic-Tac-Toe, Nim, and even obscure "games" such as the Prisoner's Dilemma. Game theory has even branched out to scenarios that most people would not consider a "game." Mathematicians have numerous ways of determining the winning strategies in these "games."

One classification of games that is very important is "combinatorial games." These games have multiple characteristics, such as they cannot end in a draw and each player alternates taking turns. One way of deciding who will win a combinatorial game is by creating a game tree and labeling it with "P-positions" and "N-positions." P-positions and N-positions are categories for each "position" in a game. A P-position is a position where the previous person has a winning strategy, while an N-position is a position where the next person has a winning strategy. A huge result of this categorizing of positions is that any time a player can move to a P-position, the current position is an N-position. Consequently, this means that any time a player cannot move to a P-position, they are currently at a P-position (Friedman). Using these two statements, we can create a game tree for a game to find the winner. However, in Tic-Tac-Toe, a game that ends in a draw is not only possible, but it is common. Normally, this means that we cannot use P-positions and N-positions when discussing Tic-Tac-Toe, but in **Section 1.3**, we discover why we can use a similar labelling of positions to create a game tree to determine who wins.

1.2. Game Trees

A game tree is a mapping of all the possible outcomes of a game, where the beginning game position branches into all the possible positions that the first move can create, and it continues until no more moves can be made. **Figure 1-1** is a diagram of a game tree for a game called “The Subtraction Game”. In this version of the game, there is a pile of beans, and each player takes turns removing either two or three beans from the pile. So, if the pile starts with six beans, the next player’s option is to take away two beans, leaving four in the pile, or three beans, leaving three in the pile. This can be seen in the game tree, because the position representing six beans has two arrows protruding from it, one going to the four, and one going to the three, depicting the two options the player has. The P-positions have been shaded, while the N-positions are unshaded. In order to determine which positions are P and which are N without the diagram, we would start with all of the end positions of the game. The end positions are any positions where the next player has no possible moves. Since every end position cannot move to a P-position, they must be P-positions. That means the end positions for this game, zero beans and one bean, are P-positions. Then, we look at each pile of beans that can lead to a pile of zero beans and one bean and label them as N-positions. We continue to use the two statements explained in **Section 1.1** to create the entire game tree for this Subtraction game. We can use this diagram to determine that when the game starts in a P-position, the second player will win, and when the game starts on an N-

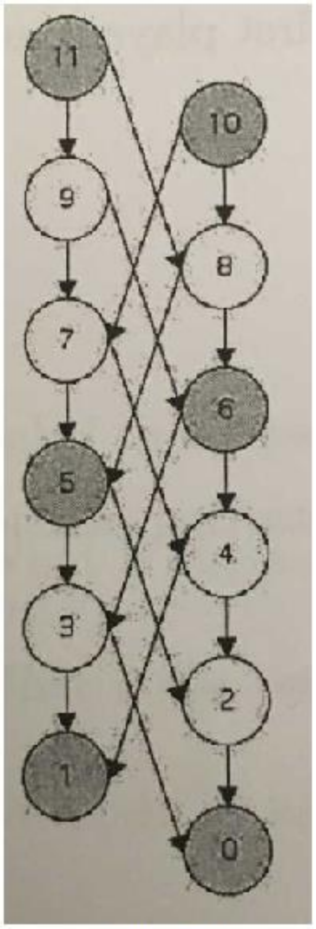


Figure 1-1: Subtraction Game Game Tree

position, the first player will win. So, if there is a pile of 8 beans, we can see that the first player will win this game.

1.3. First Player Advantage

Before we look at a game of Tic-Tac-Toe, we must also learn about the first player advantage. In many games, the first player has an inherent advantage, and some of this can be explained by the "strategy stealing argument." The strategy stealing argument is used for games like Tic-Tac-Toe, where both players have the same options, and each move does not put that player at an inherent disadvantage. If we assume that the second player has a winning strategy, the first player can take that winning strategy after making a move that does not put him at a disadvantage and does not prevent him from taking the strategy. Thus, if the second player has a way to win in a game like Tic-Tac-Toe, the first player can steal the win (József). Since the second player technically has no way of winning, we can consider the second player forcing a draw as the second player "winning" the game. This changes how we create the game tree. Instead of labeling each position as N and P, we can label them as "F" and "S", where F means the first player can win and S means the second player can draw. This causes some major differences in the game tree, which can be seen in the next chapter.

CHAPTER 2: Tic-Tac-Toe

2.1. The Original Game

Now that we understand the basics behind game theory, we can look at a regular game of 3x3 Tic-Tac-Toe. Since the second player cannot win, we will focus on whether or not the second player can prevent the first player from winning. Thus, a draw is viewed as a second player win for the purposes of an original game of Tic-Tac-Toe. One way to figure out who wins a game of regular Tic-Tac-Toe is by creating a game tree, however there are hundreds of thousands of possible outcomes for the game. **Figure 2-1** shows a game tree for Tic-Tac-Toe that only contains the first few positions, and the number of branches we would create with each new move grows exponentially. To make the game tree much more manageable, we can put a restriction on how each player chooses his next move.

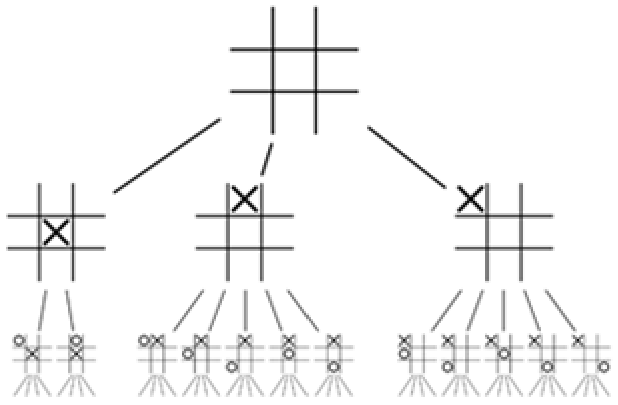


Figure 2-1: Tic-Tac-Toe Game Tree

We can restrict each player to prefer placing their X or O in a winning spot, over any other spot. We can also force each player to prefer blocking an opponent’s winning move if that player does not have a winning move. The last thing we can do to limit the game tree is by ignoring every position that is symmetrically equivalent to another position. Now, instead of there being nine original moves that the first person can make, there are three: a corner space, an edge space, and the middle space. **Figure 2-2** shows one branch of this game tree, with each position labelled in the top right corner as either an S or F position. As we can see, the branch ends in an S because the second player creates a draw in this scenario. With P and N positions, the label depends on what each position can move to, as discussed in Chapter 1, but when we use F and S, determining which label to use becomes more complicated. We can define an F-position

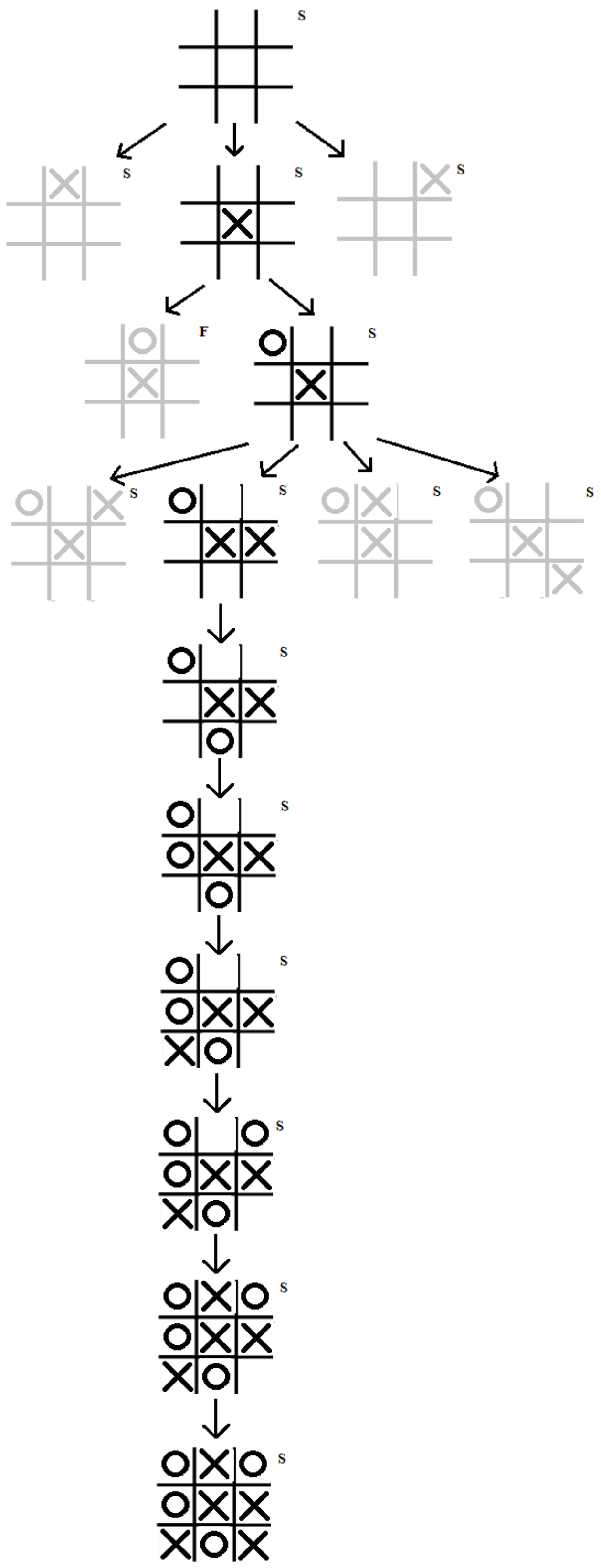


Figure 2-2: Tic-Tac-Toe Game Tree

as any position where the first player has won, the first player can move to an F-position, or the second player can only move to F-positions. Likewise, an S-position is any position where the second player has forced a draw, the second player can move to an S-position, or the first player can only move to S-positions. So, we can label the bottom of the branch as an S-position, because the first player did not win the game. We can use these new definitions to determine that this branch leads to a second player draw. The entire game tree ends with ninety-five branches that a game of Tic-Tac-Toe can take, where eighty-three of those paths end in a draw, ten of them result in the first player winning, and two paths end in the second player winning. Using this game tree, we discover that if the first player places an “X” in the middle square, the optimal move for the second player is to place an “O” in a corner square. Similarly, if the first player takes either a corner or edge piece as his first move, the second player should take the middle square. This ensures that the second player will have a way of forcing a draw.

2.2. Additional Square

Before we add more rules to the game, we should first look at what happens when we change the board. The first thing we can look at is what happens when a square is added to the board. This simplifies the game a lot. If we look at a board with an extra square next to an edge square, there is a clear strategy for the first player to win. If he places an X as seen in **Figure 2-3**, the second player is forced to place an O in the middle row. If the second player does not place an O in the middle row, the first player can place an X directly to the right of his first move and now threatens a win with two separate moves. Since the second player can only prevent one of these moves, the first player will win. However, if the second player decides to place an O in the middle row, it is either in an edge square or the new square. Since both players can no longer use the new square to get three makers in a row, we can relate this game to a normal 3x3 board and look at our game tree again. If the second player takes an edge square, the game tree shows that the first player will win. If the second player took the new square, it is as if the second player never placed an O during his first turn on a 3x3 board, which is strictly worse than taking an edge square, so the first player will still win. Thus the first player has a definitive winning strategy on this board.

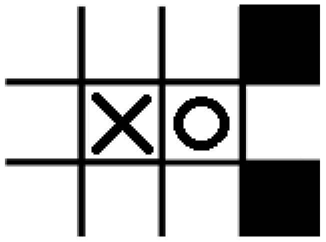


Figure 2-3: Extra Side

There are two other places where we can add an extra square. **Figure 2-4** shows a board with an extra square adjacent to a corner square. As we can see in the figure, the first player's best first move is in the square in the first row and third column. Similarly to the board with an extra square

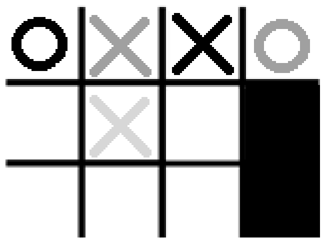


Figure 2-4: Extra Corner Side

next to an edge piece, the second player's options are very limited. In order to avoid the same situation as **Figure 2-3**, the second player is forced to move in the corner. The first player can then follow the moves in the figure to ensure a win.

A square can also be added diagonally adjacent to a corner, as seen in **Figure 2-5**. This square creates the least variation from an original game of Tic-Tac-Toe. No matter where the first player places his first X, the second player can place his O in a square that blocks off the corner piece, thus turning the board into a regular 3x3 board. Thus, the second player can force a draw.

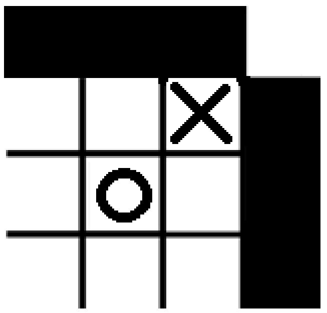


Figure 2-5: Extra Corner

2.3. 5x5 Game Board

Instead of adding single squares, we can now look at how the game changes when we add entire rows and columns to the game. For example, we can figure out who will win a Tic-Tac-Toe board with five rows and five columns, where each player's goal is to make a line using five of his markers instead of three. We can see that, since there are five rows, five columns, and two diagonals, there are twelve different ways to get five markers in a row. A game tree becomes even more impractical. Instead of creating a game tree, we can try to create a strategy the second player can enjoy that will always prevent the first player from winning. One strategy we can use is a pairing strategy, where each position has a paired position, and if the first player takes one of these positions, the second player will take its pair. **Figure 2-6** shows one possible pairing strategy the second player can use. The reason this works is because every row, column, and diagonal has a complete pair of positions. That means that since there are twelve ways to get five in a row, we need twelve pairs,

1	5	1	6	2
8	9	10	11	8
4	12		12	2
7	11	10	9	7
4	5	3	6	3

Figure 2-6: 5x5 Pairing

or twenty-four squares, to create the pairing strategy. Since a five by five grid has twenty-five squares, we have one extra space that does not need a pair, which is why the middle square in the diagram is blank. We can create an equation to determine if a grid has enough spaces to use a pairing strategy. For any board of size *s* by *s*, there are s^2 squares available. The number of pairs we need is $s+s+2$, because we need one pair for each of the *s* rows, each of the *s* columns, and the two diagonals. Since we need two squares for each pair, we can create the following inequality to determine if the board has enough squares for a pairing strategy.

$$(1) \quad s^2 = 2(2s+2) = 4s+4$$

Since this equation holds true for any integer greater than four, the second player has enough empty squares for a pairing strategy for all square boards of size 5 by 5 or greater.

2.4. Larger than 5x5 Board

Although there are enough spaces to create a pairing strategy for boards of size 6 by 6 and greater, this does not necessarily mean that we can use a pairing strategy. Proving that there is at least one existing pairing strategy for these boards requires a more rigorous proof. To prove this, we must first create labels for different aspects of a Tic-Tac-Toe board. First, let us define each square individually. Let a square be represented by a pair of coordinates (x, y), where “x” is the row of the square, and “y” is the column of the square. Thus, the square in the 3rd row and 4th column of a grid can be represented by the coordinate (3, 4). Next, let us create a symbol for representing each pair in the pairing strategy. Let (a, b) and (c, d) be the two squares used in a pairing strategy. We can represent this by writing a “&” between them, such that (a, b) & (c, d) is a pair. Let “*s*” be used to represent the number of rows and columns in a board of size *s* by *s*. Since there are *s* rows and columns, we can represent these as R_n and C_n, where “n” is the specified row or column number. There are also two diagonals, so we can define D_1 as the descending diagonal from left to right, and D_2 as the ascending diagonal from left to right. Now,

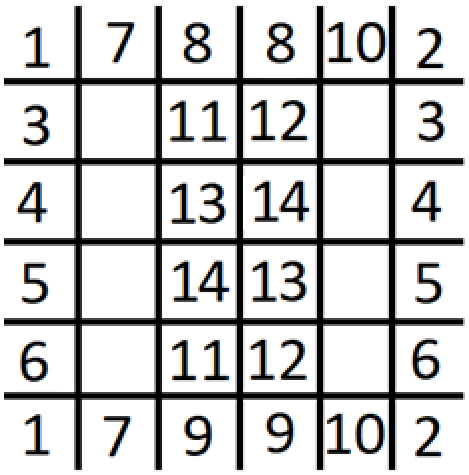
$$R_n = \begin{cases} (n,1)\&(n,s), & 1 < n < s \\ (1,\frac{s}{2})\&(1,\frac{s}{2}+1) & \\ (s,\frac{s}{2})\&(s,\frac{s}{2}+1), & s = \text{even} \\ (1,\frac{s-1}{2})\&(1,\frac{s+3}{2}) & \\ (s,\frac{s-1}{2})\&(s,\frac{s+3}{2}), & s = \text{odd} \end{cases}$$

we can create the equations.

$$C_n = \begin{cases} (1,1)\&(s,1), & n = 1 \\ (1,s)\&(s,s), & n = s \\ (2,n)\&(s-1,n), & n = \frac{s-1}{2}, \frac{s}{2}, \frac{s}{2}+1, \frac{s+3}{2} \\ (1,n)\&(n,n), & n \neq 1, \frac{s-1}{2}, \frac{s}{2}, \frac{s}{2}+1, \frac{s+3}{2}, s \end{cases}$$

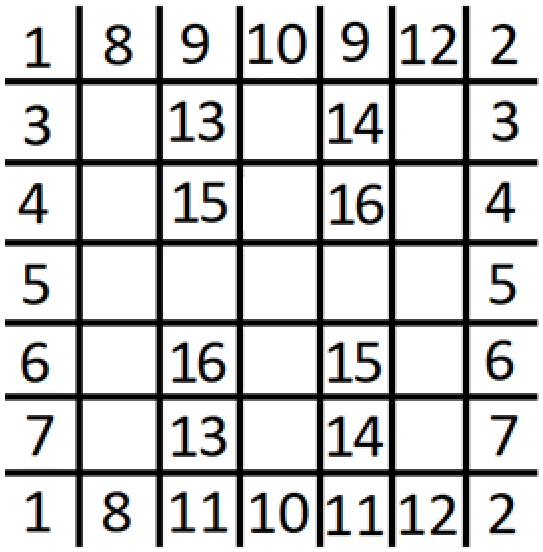
$$D = \begin{cases} (\frac{s}{2},\frac{s}{2})\&(\frac{s}{2}+1,\frac{s}{2}+1) & \\ (\frac{s}{2},\frac{s}{2}+1)\&(\frac{s}{2}+1,\frac{s}{2}), & s = \text{even} \\ (\frac{s-1}{2},\frac{s-1}{2})\&(\frac{s+3}{2},\frac{s+3}{2}) & \\ (\frac{s-1}{2},\frac{s+3}{2})\&(\frac{s+3}{2},\frac{s-1}{2}), & s = \text{odd} \end{cases}$$

So, for an 8x8 board, the diagonal pairs would be (4, 4)&(5, 5) and (4, 5)&(5, 4) using these equations. Since each of these squares is unique for boards of size 6 by 6 or greater, these equations can be used to create a pairing strategy for these boards. **Figures 2-7** and **2-8** show the pairing strategies for a 6x6 and 7x7 board using these equations. Similarly to the 5x5 pairing strategy, these grids also have empty squares. However, the number of squares differs. We can actually calculate this by using equation (1). The difference between the left side and right side of the equation calculates the number of empty squares that should appear on the grid. So, for a 6x6 grid, since the difference between thirty-six and twenty-eight is eight, there should be eight empty squares on the grid, which can be seen in the figure.



1	7	8	8	10	2
3		11	12		3
4		13	14		4
5		14	13		5
6		11	12		6
1	7	9	9	10	2

Figure 2-7: 6x6 Pairing



1	8	9	10	9	12	2
3		13		14		3
4		15		16		4
5						5
6		16		15		6
7		13		14		7
1	8	11	10	11	12	2

Figure 2-8: 7x7 Pairing

2.5. 4x4 Game Board

Now we need to look at the square board of size four and discover a new way to figure out who wins this game. Since the board requires ten pairs for the pairing strategy, but there are only sixteen squares, the second player cannot simply use a pairing strategy, however we can force the second player to make specific choices during his first two moves, as seen in **Figure 2-9**. The reason this set of moves is important is because no matter what the first player's first two moves are, the second player can always move to some rotation of this figure. Now there are only five options for four markers in a row, and fourteen empty spaces (the sixteen original spaces minus the two O's). Now the pairing strategy is viable, but there is a huge problem. We have no way to predict the first player's first three moves. In order to create a

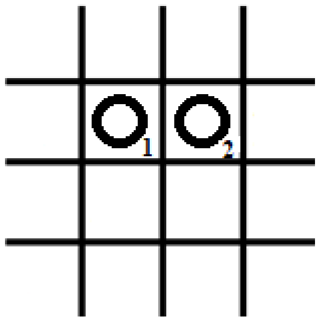


Figure 2-9: First Two Moves

successful pairing strategy, we must create a different pairing strategy for each possible combination of squares the first player decides to take. This is where the unused spaces become important. Since we need to make five pairs, and there are fourteen empty spaces, we can look at the options for the four unused squares to create grids. **Figure 2-10** shows one such grid, where the unused squares are marked “B” for “Blank”. We can then construct a pairing grid with these four blank squares, as seen in **Figure 2-11**. Of course, whether or not we can use this grid depends on the first player’s first three moves, since after our second “O”, the first player will make a total of three moves. Any grid using the blank square setup can only be used when the first player has taken at least two of the blank squares as his first three moves. Now, we must create a grid, using this blank square strategy, for every possible first three moves that the first player can make.

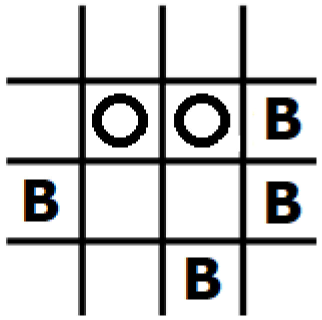


Figure 2-10:
Blank Squares

Similarly to our game tree for the 3x3 board, we can use symmetry to erase most of the possibilities. Instead we can split up his choices into four categories: the first three moves can either contain two or more edges, the two remaining middle squares, two or more corners, or a corner, a middle square, and an edge square. The following page shows the pairing strategies for each category. Note that the last pairing strategy shown is the same as the “2 Middle” grid. Thus, the second player can create a draw in a 4x4 game of Tic-Tac-Toe if that player follows the nine grids seen in **Figure 2-12**.

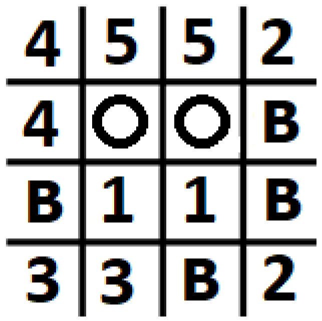


Figure 2-11:
Blank Square
Pairing Grid

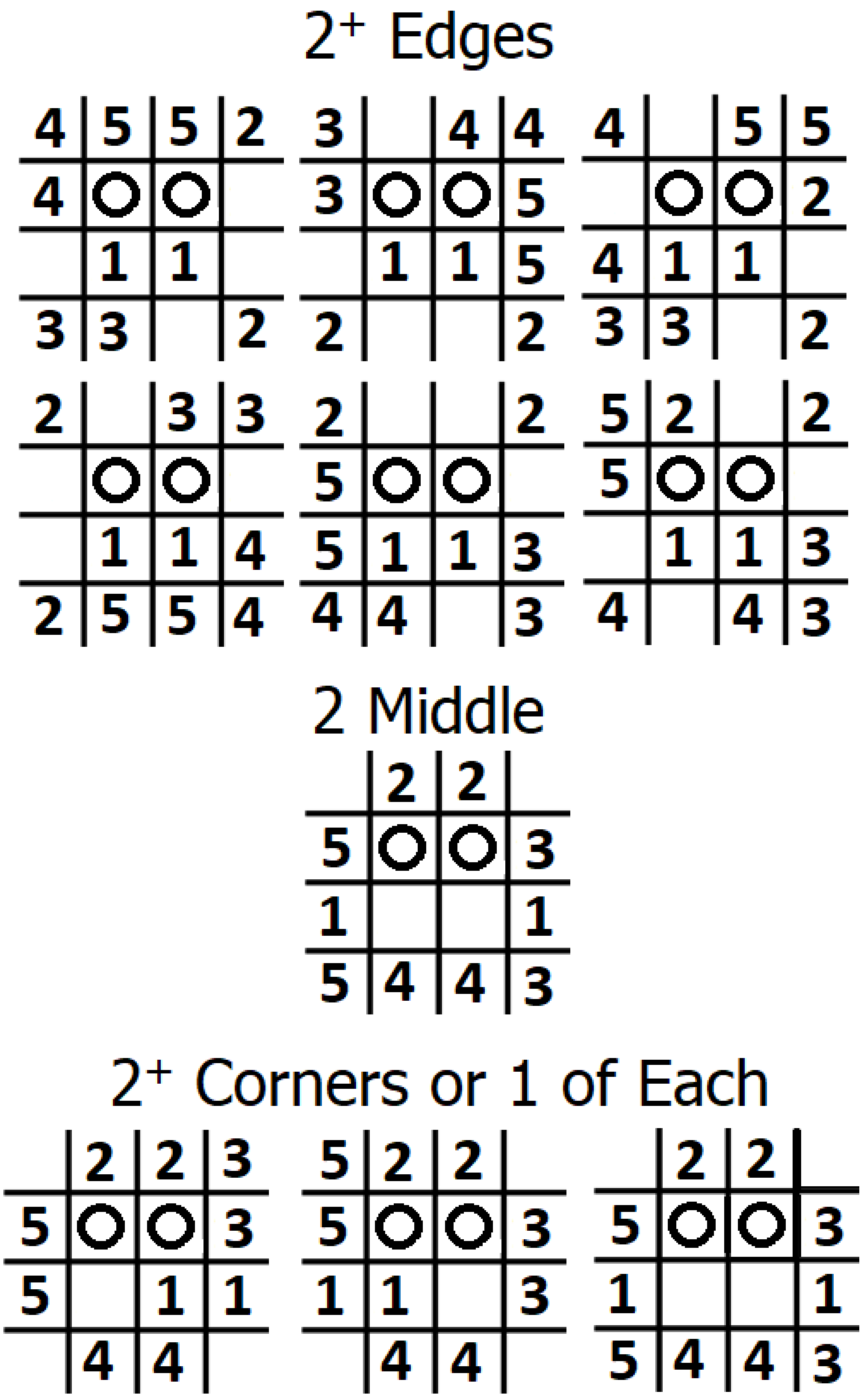


Figure 2-12: 4x4 Pairing Grids

CHAPTER 3: 3D Tic-Tac-Toe

3.1. Adding the Third Dimension

The next step after adding rows and columns, is adding dimensions. This means we can look at a board that is 3x3x3. The rules of the game are very similar to regular Tic-Tac-Toe. Each player takes turns placing an X or O in an empty square, and the first player to get three X's or O's in a row, column or diagonal wins, but now the player can also get three in a row through the third dimension, either in a vertical column or a diagonal line through the three layers. For simplicity, we will separate the 3x3x3 board into three separate boards, labelling each layer from one to three, with the middle layer as layer two. Just like in 3x3 Tic-Tac-Toe, the best move for the first player is in the middle square of the board, which would be the middle square of layer two. This is because the middle square gives the first player thirteen ways to get three in a row, while a corner square, a middle square, and an edge square in layer one or three give five, five, and four ways to win respectively. The edge and corner squares of layer two only give three and four ways to win respectively.

After the first player plays in the middle square, the second player's moves can be put into three different categories: the second player will either play in the middle of another layer, in an edge or corner of another layer, or in an edge or corner in layer two. Without loss of generality, we can assume that if the second player chooses to start on either layer one or layer three, he will start on layer one. If the second player responds in the middle of layer one, the first player should respond by playing in layer one. This threatens a diagonal through the layers, so the second player must be respond in the opposite square of layer three to stop the first player from winning. Now, the first player should place his X adjacent to the X he placed on layer one. Now, the first player threatens two ways to win, and the second player can only stop one of these, so the first player wins. The second player also has the option to play in an edge or corner of layer one instead. If he does, the first player should respond in the middle of that layer. This threatens a vertical column through the middle of the layers, so the second player has to respond in the middle square of layer three. Now, if the first player places an X in any empty square on layer one, he threatens two ways of winning, and has won the game. The last possible first move for the second player is in layer two. If that happens, the first player should use the same strategy as the previous case, by taking the middle square of layer one and then an edge or corner square in layer one to win. So, the game of 3x3x3 Tic-Tac-Toe is a first player win.

3.2. Otrio

In 2015, Brady Peterson designed the game Otrio, which was marketed as “Next Level Tic-Tac-Toe.” In this game, each player has nine circles, with three large, three medium, and three small circles. These circles are colored to differentiate which player used which circle. The board is similar to a Tic-Tac-Toe board, however once a circle has been placed in a square, the other two sizes of circles are still able to be placed in that square. The winner of Otrio is the first player to get three circles of the same size in a row, column, or diagonal; all three sizes of circles in one square; or one of each sized circle in a row, column, or diagonal in either ascending or descending order. This game can be played with two to four players, but we will only look at the strategy for two players.

This game is very similar to 3D Tic-Tac-Toe, if we view each circle size as one of the “layers” in 3D Tic-Tac-Toe. This means the only difference between Otrio and 3D Tic-Tac-Toe fundamentally is that each player is limited to three circles of each size, which in 3D Tic-Tac-Toe would be equivalent to saying each player can only place three markers in each layer. Since our strategy for 3D Tic-Tac-Toe never required the first player to place three markers in a single layer, the same strategy can be applied to Otrio for a first player win. We just need to adjust the strategy so it handles circle sizes instead of layers. So, the first player should start with the medium size circle in the middle square. Similar to 3D Tic-Tac-Toe, we can assume that the second player if the second player chooses either the small or large circle, he will choose to use the small circle. If the second player places a small circle in an edge or corner, the first player should respond with the small circle in the middle square. Now, the first player has a medium and small circle in the middle square, so the second player must place a large circle in the middle square. The first player can place another small circle in any empty edge or corner square and threaten two ways of winning. Alternatively, the second player's first move could be a small circle in the middle square. Then, the first player should respond with a small circle in any edge or corner square. The second player must place a large circle opposite this small circle to stop the first player from winning. The first player should place a small circle adjacent to his previous small circle to threaten to ways of winning. The last option the second player has for his first move is to place a medium size square in an edge or corner. The first player should place a small circle in that same square. The second player must place a large circle opposite this small circle. Now, the first player can threaten two ways of winning by placing a small circle in the same square that the second player just played in. So, just like 3D Tic-Tac-Toe, Otrio is a first player win.

CHAPTER 4: Technology

4.1. MATLAB

Now that we have looked at varying sizes for Tic-Tac-Toe boards and how they impact the winner of the game, we can start changing the rules of Tic-Tac-Toe. However, changing the rules of the game makes finding the winning strategy much more difficult to do without a computer. One of the big reasons for this is that the best first move is not apparent. In regular Tic-Tac-Toe, the middle squares are always the best squares because they create the most opportunities for getting three markers in a row, but when the rules of the game starts changing, this is not necessarily the best first move. Another important feature of normal Tic-Tac-Toe is that we considered a tie to be a second player win, because of the first player advantage, but this advantage is not applicable to some of the variations. In order to handle these changes in the game's characteristics, the best way to figure out what the winning strategies for these variations is analyzing a game tree, but just like in regular Tic-Tac-Toe, there is a very high number of branches in each of these game trees, so it becomes necessary to use a computer program to create these game trees. One program that works well for creating these game trees is MATLAB, so that is what we will use when looking at some of the other variants of Tic-Tac-Toe.

The first thing we can do with MATLAB is create a game tree for the regular 3x3 game of Tic-Tac-Toe. Instead of using X's and O's, we can label the first player markers as 1's and the second player's markers as -1's. We can use this to create a game tree for the 3x3 Tic-Tac-Toe game. The game tree is stored as a matrix of 3x3 matrices, where each of those 3x3 matrices represents a board position. The row of the game tree matrix signifies which game of Tic-Tac-Toe we are looking at, and the column represents the turn number. The program starts in the top left of the board and moves down the column, then to the next column, looking for the first empty square to place a marker. So, the 1st row and column of the game tree matrix contains a 3x3 matrix with all 0's, and the 1st row and 2nd column contains a 3x3 matrix with a 1 in the top left and 0’s in the rest of the matrix. Since the top left element of the matrix has already been filled, the program will now place a -1 in the top middle element for the next 3x3 matrix. This continues until there are three of the same marker in a row, column, or diagonal, or all nine elements of the matrix have been filled. Then the program looks at the second last turn of that game and chooses a different square for that turn’s marker. If all possible moves have been examined, the program will move back another turn.

This program does not eliminate rotationally symmetric boards, so the number of branches in the game tree will be much larger than in the previous chapter. Now, there are 255,168 end positions for the game, with 131,184 games where the first player wins and 123,984 games where either the second player wins or the game ends in a tie. We can then create a program that uses the F and S positions from earlier to create a separate matrix containing the winning player for each of the positions, whether they are end positions or not. This matrix labels each game as 1 or -1, with 1 representing an F position and -1 representing an S position. This gives us a total of 549,946 positions, with 269,056 F positions and 280,890 S positions. We can then look at the beginning of the game and see that this is an S position, confirming the results of the reduced game tree.

4.2. Misère

Since the program for solving the original game of Tic-Tac-Toe works, we can solve some of the variations of Tic-Tac-toe by editing the code from this program. For example, we can solve the winner of the Misère variation by making a few changes to the code. One of the alterations is, of course, switching the F and S positions of a game because a first player win now becomes a second player win. An even more important change, however, is that now a tie game cannot be considered a second player win. Since the first player advantage is no longer applicable to this game, the second player could have a winning strategy. So, we need to edit the code to separate second player wins from draws. This means we must now create a third type of position. So, we now have F, S, and D positions. This does not change the definitions for F and S positions, so F positions are still positions where it is either the first player's turn and he can move to an F position or it is the second player's turn and he can only move to F positions. However, if the second player cannot move to an S position, but can move to a D position, and similarly if the first player cannot move to an F position, but can move to a D position, the position is a D position. Once we have done this, as well as a few minor other changes, we can see that there are still 549,946 positions, since we did not change the possible scenarios that can happen in the game, but the number of F and S positions has changed. Now, there are 103,160 F positions, 322,848 S positions, and 123,938 D positions. We can then check the winner at the start of the game and see that it is a "draw" position. This means that if both players understand the best strategies in Misère Tic-Tac-Toe, they will draw every time.

Now that we know the winner of this variation, we need to analyze the game tree that the program gives us to see what strategy the players should follow. When we look at each of the nine positions for the first X, we can see that eight of these lead to S positions, while one leads to a draw. This drawing strategy is the middle square for the first move, which is very surprising because in order to both win and lose a game of Tic-Tac-Toe, the best first move does not change. After the first player takes the middle square, the player needs to use a mirror strategy. So, if the second player goes in the top right, the first player should move in the bottom left, and if the second player goes in the middle left, the first player should respond in the middle right. This strategy eliminates the possibility of getting three X's in a row that include the middle square. Not only that, but it also stops the first player from getting a three in a row anywhere, because if there are three X's in the bottom row, this means that there must have been three O's in the top row the turn before this one, so the first player would have already won.

When the first player takes an edge or a corner square as the first move, the strategy for the second player to win is not as simple. There are three things the second player needs to avoid: letting the first player create a mirror strategy, blocking all of the first player's losing squares, and losing the game. One of the best strategies for the second player to follow is to never take the middle square. It is still possible to win without doing so, but this limits the possibility of the first player preventing the second player from winning. Instead, the best first move is to take a square clockwise or counter-clockwise from the opposite square that the first player takes. So, if the first player goes in the top left square, the second player should respond in the middle right or bottom middle square. This is the best way to avoid those three mistakes. The reason the second player cannot go opposite the first player, is that the first player can then take the middle square and is now three moves into his mirror strategy. After this first move, the second player just needs to avoid the first and third mistake and try to avoid the second mistake as much as possible.

4.3. Veto Tic-Tac-Toe

Another version of Tic-Tac-Toe that does not require too much change to the regular Tic-Tac-Toe code is a variation called Veto Tic-Tac-Toe. In this variation, on each player's turn, the opponent can veto one move. So, if the first player takes the middle square turn one, the second player can veto that move, and the first player now has to go in another empty square. Each player can veto up to once each turn. This changes which positions are labelled as F and S. Instead of requiring the first person to be able to move to one F position on his turn, he now needs to be able to move to two F positions in order for that position to be an F position, since the first F position can be vetoed. This also alters the end positions. If eight of the nine squares are filled up, the first player cannot move in the remaining spot, because it will be vetoed. Thus, draws are also a possibility in this variation as well. Since the way to win Tic-Tac-Toe has not been altered, we can still use the first player advantage to assume that if the game is not a first player win, it is a draw game. So, we can once again lump second player wins and draws into the same category of S position. These are the only differences we need to look at while building the game tree.

MATLAB tells us that there are 422,074 positions, with 61,968 F positions and 360,106 S positions. Just like the original game, the starting position is an S position. Once again, we can look at the matrix of winners to determine what strategies each player should employ. In regular Tic-Tac-Toe, in order for a player to win, that player must threaten at least two ways to get Tic-Tac-Toe on his turn. For Veto Tic-Tac-Toe, the player now needs to threaten at least three ways to get Tic-Tac-Toe, which is much harder to do. This requires the first player to take the middle square, a corner square, and one of the edges adjacent to both of these squares, so that he threatens a diagonal, a row, and a column. This means that the second player needs to prevent the first player from taking the middle square until the first player is unable to create a position that threatens three wins. Since the second player will veto the first move if it is in the middle square, the first player now needs to take an edge or a corner square. In either case, the second player should take a square adjacent to the first move. If the first move was an edge, the first player can no longer create the structure needed to threaten three wins that contains his first move, so he needs to start over in a different part of the board. If the first move is a corner, the second player should veto the middle square again for the second turn. Now, the only way for the first player to create the winning structure is by taking the edge adjacent to his corner that is still empty. This means the first player has two X's in a row. The first player now has the choice to either veto the middle square or veto the square that blocks his two X's in a row. Either way, the structure can no longer threaten three ways to win, so the second player can force a tie.

4.4. Notakto

Notakto is a variation of Veto Tic-Tac-Toe where both players use an X. This means that if the first player places two X's in a row, and the second player places a third X to complete the row, this counts as the game ending, and the first player would win. This means we can use the same code as the Veto Tic-Tac-Toe code, with two differences. Instead of using 1's to represent X's and -1's to represent O's, we would only use 1's, because both players use X. Also, ties are not possible in this version of Tic-Tac-Toe, because the board can never be completely filled with X's without one of the players winning. This severely limits the number of possible board states. One of the reasons the number of board states decreases is because the game will always end by turn 7, since any time there are seven X's on a 3x3 board, there is at least one row, column, or diagonal with three X's. So, the game tree matrix for our MATLAB code is a matrix that is 23232 by 8.

When we run the MATLAB program, we see that there are 34,560 positions. 23,952 of these positions are F-positions and 10,608 of these positions are S-positions. The program also tells us that the original board state is an F-position, so Notakto is a first player win game. Now, we must figure out what the winning strategy is. The first thing we can do is look at all the positions where any move causes the last person to move to lose. This is shown in **Figure 4-1**.

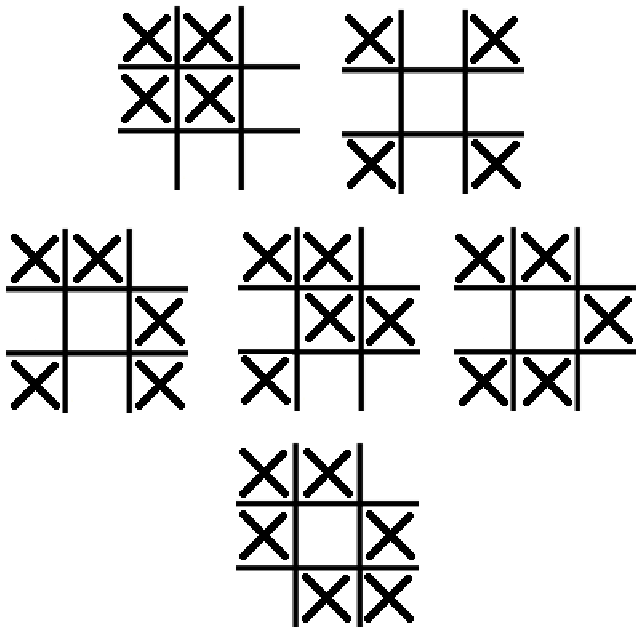


Figure 4-1: Notakto End Positions

So, there are six different positions where one more X will cause the game to end. This is important to look at because the top and bottom row have an even number of X's. This means the game will end with an odd number of X's, which means the first player loses the game. In order for the first player to win the game, his strategy is to force the game to become one of the positions in the middle row. We can determine that the best move for the first player is the middle square, because this eliminates two of the three positions where the second player wins. Now the first player just needs to avoid creating a 2x2 square of X's on the board. The best way for the first player to do this is by placing an X opposite to a square that is adjacent to the second player's last move, similar to how a knight moves in chess. So, if the second player plays in the top right, the opposite adjacent squares are the bottom middle and middle left squares. This eliminates all of the ways for the second player to win, but we can also make the strategy simpler. After the first player takes the center square, if he responds with a "knight move" to each move the second player makes, while avoiding getting three X's in a row, the first player can always force the game to look like **Figure 4-2**. While this strategy is not necessary to win on one board, Notakto is normally played on multiple boards. In order to play on multiple boards, we need to look at the new rules of the game. Each turn, the player can play on any of the boards, but once there are three X's in a row, column, or diagonal on a board, neither player can play on that board. So, the last player to win the last board loses the game.

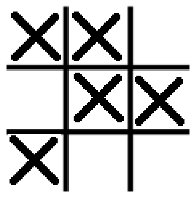


Figure 4-2: Notakto Knight Move

Now, we can look at Notakto on two boards. Since each individual game ends by turn seven, two games will end by turn fourteen, and since there are 23,232 branches in the individual games, we can square that number to get the number of branches for Notakto with two boards. This means the game tree matrix is 539,725,824 by 15. This is too large for MATLAB to handle, so we need to analyze this game without technology. One strategy that is common in Game Theory with multiple boards is the "copy-cat" strategy. With this strategy, wherever the first player plays, the second player responds in the same position on another board. However, in

Notakto with two boards, if the second player does this, the second player will win the last board, and lose the game. Instead, the second player can employ the copy-cat strategy until the first player creates one of the “end positions” that we looked at earlier. Once this happens, the second player should make three in a row on the other board, and the first player is forced to lose. This strategy works in most games, however if the first player’s first two moves are in opposite corners, the first player can end the board after the second player copies him. Now the first player can force the second player to win the remaining board, since the middle square is no longer available for the second player to take. This same problem arises when two of the first player’s first four moves are opposite corners. So, we must come up with a different strategy for the second player.

Since the middle square gives the first player control over one board, we can look at what happens when the second player tries to take the middle square on two boards. This means that whatever the first player chooses as his first move, the second player can respond in the middle square of the unused board. This means there are three cases: the first player’s first move was in a middle square, a corner square, or an edge square. If the first player’s move was in a middle square, this means wherever the first player goes for his second move, the second player can respond by winning that board. Now, we are left with one board with the middle square already taken. This means this game is now equivalent to a single board of Notakto where the second player moved first and took the middle square. So, if the first player’s first move is in the middle square, and the second player responds in the other middle square, this game is a second player win. If the first player instead takes a corner square as his first move and the second player takes the other middle square, there are a limited number of options for the first player’s next move. This first player cannot take a square in the same row or column as his first move, because the second player will win that board, and we will have the same situation as before. So, the first player can either play in an adjacent edge or an opposite corner on the same board as his first move or he can play in the board that the second player has played. If he does the former, the second player can respond in either case to create the board position in **Figure 4-3,** which we will call the “Setup” board. Now, if the first player responds in this board again, the second player can end that board, so the first player is forced to play in the other board. Now, the second player can force the first player to win that board by using the knight move. If the first player at any point plays in the Setup board, the second player can win that game and use the knight move to force the first player to win that game. So, once the first player has won the other board, the second player can play in the top right corner of the Setup board, to force the first player to win the board in two moves. Thus, the second player will win that game. A similar scenario happens if the first player takes an edge square as his first move instead of the corner square, because his next move is limited to either of the opposite edges. Thus, the second player can create the Setup board again to win the game. Thus, if the second player responds in the middle square of the empty board for a game of Notakto with two boards, the second player can win the game. So, Notakto is a first player win with one board, and a second player win with two boards.

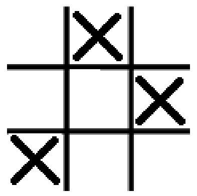


Figure 4-3: Notakto Setup

CHAPTER 5: Summary of Results

An original game of Tic-Tac-Toe has two players play on a 3x3 board, where the first player and second player take turns placing an “X” or “O” respectively on the board. The game ends either when one player wins by placing three markers in a row, either horizontally, vertically, or diagonally, or when all nine squares are filled, which causes a draw if nobody has already won. We used a game tree, which is a mapping of all the possible board positions in the game as well as which board positions lead to other board positions; as well as a system for labelling each position, to determine which player won the game. We created a game tree by manually by limiting the board positions we looked at by removing board positions that are symmetric and by limiting each player’s moves so they did not make a move that is clearly worse than another potential move. We also created a program through MATLAB that outputs the entire game tree. Both of these game trees were analyzed to show that Tic-Tac-Toe is a game where the second player could always force a tie.

Next, we looked at adding squares to the 3x3 board of Tic-Tac-Toe. The three additional squares we added are an extra side square, an extra corner square, and an extra corner side square. With each of these additional squares, the first player’s best first move was to play in the same row, column, or diagonal as the added square. When an extra side square is added, as well as when an extra side corner square is added, the first player was able to make the new square unusable, while turning the remaining 3x3 board into a position where the first player would win. However, adding an extra corner square did not affect the winner of the game. So, adding an extra side or corner side square created a first player win, while adding an extra corner square kept the outcome of the game as a draw.

Instead of just adding individual squares, we looked at how the game changes when we add entire rows and columns to the board. We were able to use a “pairing strategy” where for every move that the first player makes, there was a square on the board paired with that move that the second player would place his marker in. To create a pairing strategy, we labelled two squares in every row, column, and main diagonal with the same number. This meant the pairing strategy blocked every row, column, and diagonal from the first player. This strategy worked for all square Tic-Tac-Toe boards that are 5x5 and bigger. However, when we looked at a 4x4 board, the same pairing strategy could not be employed. This was because a square board of size *s* x *s* must satisfy the following equation in order to create a pairing strategy:

$$s^2 = 4s + 4$$

Since a 4x4 board did not satisfy this equation, we had to create a modified pairing strategy. This modified strategy required the first player to make his first three moves and the second player to make a specific first two moves. We were then able to create ten pairing strategies that covered every option for the first player's first three moves. So, as long as the second player followed the correct pairing strategy, he could force a tie in 4x4 Tic-Tac-Toe as well.

We also looked at the affect of adding a third dimension to Tic-Tac-Toe to create a 3x3x3 board. With a third dimension, the number of ways to get three in a row increased significantly. In 3x3 Tic-Tac-Toe, there were eight ways to get three in a row, but in 3x3x3, there aere forty-five ways to get three in a row. Since there were only twenty-seven squares in 3x3x3 Tic-Tac-Toe, this means we could not use a pairing strategy. However, the only strategy the first player typically needed was to place a marker in the middle square during his first move. As long as he prevented the second player from winning and tried to get three markers in a row whenever possible, the first player was almost guaranteed a win. There was also a board game called Otrio where each player has three small, medium, and large circles. The winner of Otrio was the first player to get three circles of the same size in a row, column, or diagonal; all three sizes of circles in one square; or one of each sized circle in a row, column, or diagonal in either ascending or descending order. This was similar to 3x3x3 Tic-Tac-Toe, except we used the circles' sizes to represent the third dimension. Each player also had only three of each size circle, so the first player had to play with a specific strategy in order to win. The first player placed his medium circle in the middle square. Depending on the second player's reaction, the first player's winning strategy would change a bit, but the first player was able to win Otrio regardless of the second player's moves.

The next Tic-Tac-Toe variation we looked at was Misére Tic-Tac-Toe, where the goal of this game was to force the other player to get three markers in a row. We used MATLAB program from 3x3 Tic-Tac-Toe by making some slight alterations to the code. The MATLAB program showed us that if the first player's first move was in the middle square, the game would end in a draw, but if the first player's first move was any other square, the second player would win. So, the first player's strategy to draw was to take the middle square as his first move. After this, the first player responded in the opposite square of the second player every turn. This ensured that the first player does not get three markers in a row first.

Another variation we analyzed is Veto Tic-Tac-Toe. In this game, each turn, the opponent had the option to veto one move. If a player's move was vetoed, that player chose another empty square in which to play. In normal Tic-Tac-Toe, a player was required to threaten two ways of getting three markers in a row to win, but in this variation, the player had to threaten three ways of getting three markers in a row. We changed the MATLAB program to analyze this game, and this told us that the game always ended in a draw. This was because in order to set up three threats on the board, this required the middle square and two other squares. Since the second player's best strategy prevented the first player from playing in the middle square for the first few turns and stopped the first player's setup during those first three turns as well, the first player was not able to set up three threats on the board successfully.

The last variation we looked at was Notakto. In this game, both players used an "X" as their markers, and the game was played on any finite number of boards. Once a board had three

X's in a row on it, the board could not be used anymore. The player who finished a row, column, or diagonal of three X's on the final board loses. We altered the MATLAB code again, and the program told us that this game was a first player win. The strategy behind this stemmed from the fact that there were only six ways for the board to end as long as neither player got three markers in a row before it was unavoidable. Three of these positions resulted in the second player winning, and three of them resulted in the first player winning. The first player avoided the three "second player win positions" by placing his first X in the middle square, and then responded in the square that is opposite to a square adjacent to every move that the second player made. This was similar to how a knight moves in Chess. This resulted in a first player win.

We then extended this information to a game of Notakto with two boards. One strategy that the second player employed was to place his first marker in the middle square of the empty board. This allowed the second player to set up that board either as a first player win or as a board he could sacrifice. The second player then used knight moves to force the first player to either get three in a row on both boards, or the second player "sacrificed" a board in order to force the first player to win the final board, depending on the first player's responses. Thus, Notakto with two boards was a second player win game.

CHAPTER 6: Future Work

The last variation of Tic-Tac-Toe we looked at was Notakto. This game could be played on any finite number of boards, but during this research I only looked at playing on one or two boards. Since the game tree for two boards was too large for MATLAB to handle, the game tree for three boards will also most likely be too large for MATLAB to handle. If I were to continue analyzing Notakto with more boards, the two options I would pursue are to either use the information from the one and two board strategies and try to find patterns within them that could help with a strategy for three boards or decrease the size of the game tree the MATLAB program needs to create, like I did in Section 2.1 for 3x3 Tic-Tac-Toe. Reducing the game tree makes the number of first moves we need to look at decrease from twenty-seven to three, since it does not matter which board the first player's first marker is placed. However, the MATLAB program would need to be adjusted a lot in order to remove symmetric boards, and once I look at more than three boards, the reduced matrix will eventually be too large for MATLAB, so I would start with the first approach.

Another variation that I never analyzed in the paper was Tic-Tac-Toe Forget. In this variation, each marker disappears after it is on the board for three turns. After playing this variation for months, I have made an assumption that the game will result in neither player winning or losing if both players play optimally, so my analysis of the game will start with that assumption, until I am either proven right or the assumption breaks. Once again, I have two ways to approach finding the winning player and strategy. The first option is to find every possible arrangement of three X's and three O's where neither player can win within the next turn. If both players can prevent the other player from winning, this means that there are multiple cycles of these positions of three X's and O's where either player can force the game to continue indefinitely through these cycles while avoiding any other positions. There are a lot of positions with three X's and three O's though, especially considering that the order each marker was played is considered a different position, so this next approach would most likely be what I use to find the winner. Due to the game’s rules, there are a very limited number of ways in order for a player to win. In order to win, the winner must have two markers surrounding one of the loser’s markers, and the loser’s marker must disappear this turn, while the winner’s two markers cannot disappear this turn. I would start by marking down these positions and seeing what circumstances must arise in order for these positions to be possible. Assuming both players can prevent the other player from winning, this means there must be some contradiction among those circumstances for each position.

APPENDIX

The following is the MATLAB code for the game of Tic-Tac-Toe.

```
clear;
clc;
T{255169,10}=[];
W=zeros(255168,10);
M0=[0,0,0;0,0,0;0,0,0];
M{1,1}=[1,0,0;0,0,0;0,0,0];
M{2,1}=[0,0,0;1,0,0;0,0,0];
M{3,1}=[0,0,0;0,0,0;1,0,0];
M{1,2}=[0,1,0;0,0,0;0,0,0];
M{2,2}=[0,0,0;0,1,0;0,0,0];
M{3,2}=[0,0,0;0,0,0;0,1,0];
M{1,3}=[0,0,1;0,0,0;0,0,0];
M{2,3}=[0,0,0;0,0,1;0,0,0];
M{3,3}=[0,0,0;0,0,0;0,0,1];

n=1;t=1;
T{n,t}=M0;

for t1=1:9
i=t1;j=1;
    while i>3, i=i-3;j=j+1;
    end
    t=t+1;
    T{n,t}=M{i,j};
for t2=1:9
i=t2;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
    if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
    end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}-M{i,j};
    for t3=1:9
i=t3;j=1;t4=0;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}+M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end
    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
```

```
                if abs(c(1,1:3))~=3

        % Turn 4
        for t4=1:9
i=t4;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}-M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3

        % turn 5
            for t5=1:9
i=t5;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}+M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3

        % turn 6
            for t6=1:9
i=t6;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}-M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3

        % turn 7
            for t7=1:9
i=t7;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}+M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3

        % turn 8
            for t8=1:9
i=t8;j=1;a=0;
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}-M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3

        % turn 9
            for t9=1:9
i=t9;j=1;a=0;
```

```
    while i>3, i=i-3;j=j+1;
    end
        if numel(T{n-a,t})==0
        while numel(T{n-a,t})==0
        a=a+1;
        end
        end
    if T{n-a,t}(i,j)==0
    t=t+1;
    T{n,t}=T{n-a,t-1}+M{i,j};
    d1=0;d2=0;c=sum(T{n,t});r=sum(T{n,t},2)';
    for x=1:3
        d1=d1+T{n,t}(x,x);
        d2=d2+T{n,t}(x,4-x);
    end

    if abs(d1)~=3
        if abs(d2)~=3
            if abs(r(1,1:3))~=3
                if abs(c(1,1:3))~=3
                    W(n,t)=-1; n=n+1; t=t-1;
                else
                    for x=1:3
                        if abs(c(1,x))==3
                            W(n,t)=c(1,x)/3;
                            n=n+1;t=t-1;
                        end
                    end
                end
            else
                for x=1:3
                if abs(r(1,x))==3
                 W(n,t)=r(1,x)/3;
                 n=n+1;t=t-1;
                end
                end
            end
        elseif abs(d2)==3
            W(n,t)=d2/3;
            n=n+1; t=t-1;
        end
    elseif abs(d1)==3
        W(n,t)=d1/3;
        n=n+1; t=t-1;
    end
    end
if t9==9
    t=t-1;
end
            end
else
                    for x=1:3
                        if abs(c(1,x))==3
                            W(n,t)=c(1,x)/3;
                            n=n+1;t=t-1;
                        end
                    end
                end
            else
                for x=1:3
                if abs(r(1,x))==3
                 W(n,t)=r(1,x)/3;
                 n=n+1;t=t-1;
```

```
                    end
                    end
                end
            elseif abs(d2)==3
                W(n,t)=d2/3;
                n=n+1; t=t-1;
            end
        elseif abs(d1)==3
            W(n,t)=d1/3;
            n=n+1; t=t-1;
        end
        end
    if t8==9
        t=t-1;
    end
                end
    else
                            for x=1:3
                                if abs(c(1,x))==3
                                    W(n,t)=c(1,x)/3;
                                    n=n+1;t=t-1;
                                end
                            end
                        end
                    else
                        for x=1:3
                        if abs(r(1,x))==3
                         W(n,t)=r(1,x)/3;
                         n=n+1;t=t-1;
                        end
                        end
                    end
                elseif abs(d2)==3
                    W(n,t)=d2/3;
                    n=n+1; t=t-1;
                end
            elseif abs(d1)==3
                W(n,t)=d1/3;
                n=n+1; t=t-1;
            end
            end
        if t7==9
            t=t-1;
        end
                    end
        else
                                for x=1:3
                                    if abs(c(1,x))==3
                                        W(n,t)=c(1,x)/3;
                                        n=n+1;t=t-1;
                                    end
                                end
                            end
                        else
                            for x=1:3
                            if abs(r(1,x))==3
                             W(n,t)=r(1,x)/3;
                             n=n+1;t=t-1;
                            end
                            end
                        end
                    elseif abs(d2)==3
                        W(n,t)=d2/3;
            n=n+1; t=t-1;
        end
    elseif abs(d1)==3
        W(n,t)=d1/3;
        n=n+1; t=t-1;
    end
    end
if t6==9
    t=t-1;
end
            end
else
                    for x=1:3
                        if abs(c(1,x))==3
                            W(n,t)=c(1,x)/3;
                            n=n+1;t=t-1;
                        end
                    end
                end
            else
                for x=1:3
                if abs(r(1,x))==3
                 W(n,t)=r(1,x)/3;
                 n=n+1;t=t-1;
                end
                end
            end
        elseif abs(d2)==3
            W(n,t)=d2/3;
            n=n+1; t=t-1;
        end
    elseif abs(d1)==3
        W(n,t)=d1/3;
        n=n+1; t=t-1;
    end
    end
if t5==9
    t=t-1;
end
            end
else
                    for x=1:3
                        if abs(c(1,x))==3
                            W(n,t)=c(1,x)/3;
                            n=n+1;t=t-1;
                        end
                    end
                end
            else
                for x=1:3
                if abs(r(1,x))==3
                 W(n,t)=r(1,x)/3;
                 n=n+1;t=t-1;
                end
                end
            end
        elseif abs(d2)==3
            W(n,t)=d2/3;
            n=n+1; t=t-1;
        end
    elseif abs(d1)==3
        W(n,t)=d1/3;
        n=n+1; t=t-1;
    end
    end
if t4==9
    t=t-1;
end
        end
else
                    for x=1:3
                        if abs(c(1,x))==3
                            W(n,t)=c(1,x)/3;
                            n=n+1;t=t-1;
                        end
                    end
                end
            else
                for x=1:3
                if abs(r(1,x))==3
                 W(n,t)=r(1,x)/3;
                 n=n+1;t=t-1;
                end
                end
            end
        elseif abs(d2)==3
            W(n,t)=d2/3;
            n=n+1; t=t-1;
        end
    elseif abs(d1)==3
        W(n,t)=d1/3;
        n=n+1; t=t-1;
    end
    end
if t3==9
    t=t-1;
end
    end
    end
if t2==9
    t=t-1;
end
end
end

TT{255168,10}=0;
for x=1:255168
    for y=1:10
        if numel(T{x,y})==9
            TT{x,y}=T{x,y};
        end
    end
end


for t=10-(0:8)
    nn=n-1;
    while nn>1
        c=1;V=zeros(1,9);
        while numel(TT{nn,t-1})==0
            if numel(TT{nn,t})==9
                V(1,c)=W(nn,t);
                c=c+1;
            end
            nn=nn-1;
        end
        V(1,c)=W(nn,t);
        if mod(t,2)==0 && sum(V(:)==1)>0
            W(nn,t-1)=1;
        elseif mod(t,2)==0 && sum(V(:)==1)==0
            W(nn,t-1)=-1;
        elseif mod(t,2)==1 && sum(V(:)==-1)>0
            W(nn,t-1)=-1;
        elseif mod(t,2)==1 && sum(V(:)==-1)==0
            W(nn,t-1)=1;
        end
        nn=nn-1;
    end
end
```

REFERENCES

Friedman, E. (2011). *Introduction to Game Theory*. Stetson University, pp. 7-8

József, B. (2008). *Combinatorial Games: Tic-Tac-Toe Theory*. Encyclopedia of Mathematics and its Applications, Cambridge: Cambridge University Press, p. 74

Ibraham, Joe. “Make Tic-Tac-Toe Look Like Child’s Play with Otrio.” *The Toy Insider*, 2 Mar. 2018, www.thetoyinsider.com/otrio-game-review/