James Jiang

Modelling Tic-Tac-Toe

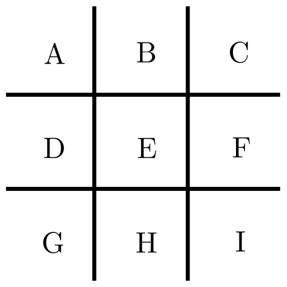
A Matrical Dissection

University of Toronto
APS162H1 S Calculus for Engineers I
Professor Shai Cohen
27/08/2017

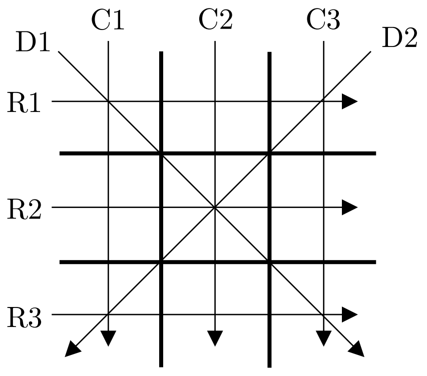
Tic-tac-toe, largely due to its simplicity, is perhaps one of the most analyzed games in history. Many algorithms have been developed to play the game perfectly (including the well-known minimax algorithm – a general algorithm in game theory that minimizes the maximum score of the opponent), and a countless number of software programs have been written to implement these algorithms. Almost always, the program will require iterating through many possible moves, sometimes 3 or more moves ahead, to determine the best choice of move for the current game board. As a result, inefficiencies can plague otherwise ingenious code.

Linear algebra can help us improve the efficiency of tic-tac-toe programs. In particular, we will see how representing every move with a specific matrix can lend us insight into the best move for every game position.

Consider a tic-tac-toe board with the allowable moves labelled A – I as follows:



We can label the rows, columns, and diagonals as well:



Hence, there are eight ways to win a game of tic-tac-toe – three ways to win on rows, three on columns, and two on diagonals. These will be referred to as *win positions*.

Consider a 1×8 matrix of the form $\begin{bmatrix} \mathrm{R1} & \mathrm{R2} & \mathrm{R3} & \mathrm{C1} & \mathrm{C2} & \mathrm{C3} & \mathrm{D1} & \mathrm{D2} \end{bmatrix}$ (the ordering of these win positions is, of course, arbitrary). Each of the terms in the matrix will take on one of two possible values, 0 or 1. For each of the allowable moves, let us assign to it a 1×8 matrix that corresponds to whether or not it contributes to a certain win position. A 1 will indicate that the move is part of the corresponding win position, while a 0 will indicate that it is not.

For example, since the move labelled D is included (and *only* included) in the R2 and C1 win positions, we assign it the matrix $\begin{bmatrix}0 & 1 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$. In the same way, the rest of the moves are assigned these matrices:

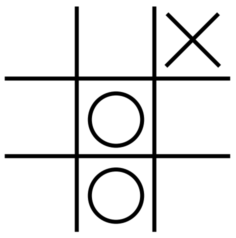
$$\begin{aligned}
\text{A} &= \begin{bmatrix}1 & 0 & 0 & 1 & 0 & 0 & 1 & 0\end{bmatrix}\\
\text{B} &= \begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}\\
\text{C} &= \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 1 & 0 & 1\end{bmatrix}\\
\text{D} &= \begin{bmatrix}0 & 1 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}\\
\text{E} &= \begin{bmatrix}0 & 1 & 0 & 0 & 1 & 0 & 1 & 1\end{bmatrix}\\
\text{F} &= \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}\\
\text{G} &= \begin{bmatrix}0 & 0 & 1 & 1 & 0 & 0 & 0 & 1\end{bmatrix}\\
\text{H} &= \begin{bmatrix}0 & 0 & 1 & 0 & 1 & 0 & 0 & 0\end{bmatrix}\\
\text{I} &= \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 1 & 1 & 0\end{bmatrix}
\end{aligned}$$

Any tic-toe-board, complete or incomplete, can be represented by a *game matrix.*

A game of tic-tac-toe will begin with the initial game matrix $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$, and proceeds in the following way:

1. Player #1 will select a move that has not been chosen yet (for their first turn, this will be all nine allowable moves). Its corresponding matrix will be *added* to the game matrix. This resulting matrix becomes the new game matrix.
2. Player #2 will then select from the list of remaining moves. Its corresponding matrix will be *subtracted* from the game matrix. Again, the result becomes the new game matrix.
3. Play proceeds until the game matrix contains either a 3 (in which case Player #1 wins) or a -3 (Player #2 wins). Consider the (impossible) final game matrix $\begin{bmatrix}0 & 0 & 3 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$. This indicates that Player #1 has won by getting 3-in-a-row on R3.

As an example, the tic-tac-toe board



can be represented as a matrix by taking the first player's moves' (O's) matrices and adding them to the initial game matrix, then taking the second player's moves' (X's) matrices and subtracting them from the resulting matrix. Notice that the order in which the moves were played does not matter (Player #1 could've gone center then side, or side then center), since addition is a commutative operation. Hence, the above position can be represented as

$$\begin{array}{rl}
\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} & \text{initial game matrix} \\
+\begin{bmatrix}0 & 1 & 0 & 0 & 1 & 0 & 1 & 1\end{bmatrix} & \text{Player \#1 moves on E} \\
+\begin{bmatrix}0 & 0 & 1 & 0 & 1 & 0 & 0 & 0\end{bmatrix} & \text{Player \#1 moves on H} \\
-\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 1 & 0 & 1\end{bmatrix} & \text{Player \#2 moves on C} \\
\hline
\begin{bmatrix}-1 & 1 & 1 & 0 & 2 & -1 & 1 & 0\end{bmatrix} & \text{game matrix}
\end{array}$$

or simply $\begin{bmatrix}-1 & 1 & 1 & 0 & 2 & -1 & 1 & 0\end{bmatrix}$.

Notice that any two move matrices have, at most, one 1 in common. We will call this property *exclusion*. In addition, we will call two or more moves *equivalent* if they have an identical effect on the game matrix, keeping in mind that the ordering of the win positions was arbitrary. For example, all corner moves are equivalent on the first move, since all four involve changing three 0's to 1's.

When a 1 gets added to a 2 or subtracted from a -2 in the game matrix, we will call this a *win*. When a 1 gets added to a -2 or subtracted from a 2 in the matrix, we will refer to this as a *block*.

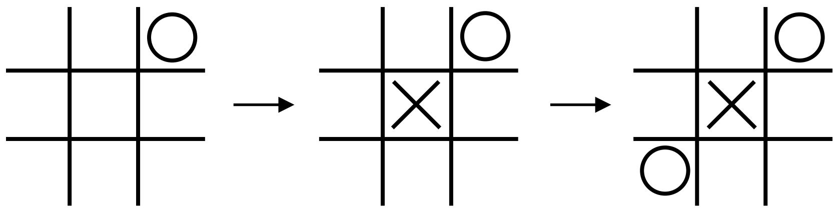
Let us examine an actual sequence of moves to develop and understand a winning strategy:

> Since the goal of Player #1 is to create a 3 in the game matrix, it is favourable for them to select a move that contains the most 1's (that is, is contained in the most win positions). Therefore, the first player should refrain from moving on the edges (which all contain only two 1's), and instead consider moving on a corner or the center. The center would of course be best, containing one more 1 than the corner moves, but the choice makes no difference against other perfect players. Suppose Player #1 moves on E – the center. The game matrix becomes $\begin{bmatrix}0 & 1 & 0 & 0 & 1 & 0 & 1 & 1\end{bmatrix}$.
>
> Player #2's intent must be to create as many -1's as possible, as well as to reverse the effect that the first player had on the game matrix to as great of an extent as possible (i.e. to turn as many 1's into 0's as they can). Upon examination, one can see that moving on a corner (all of which are equivalent in this scenario) would allow *two* 0's to become -1's and one 1 to become 0, while moving on any edge (again, all of which are equivalent) would

only allow *one* 0 to become -1 and one 1 to become 0. Thus, a corner choice is better than an edge choice. Suppose Player #2 moves on the corner C. The game matrix then becomes $\begin{bmatrix} -1 & 1 & 0 & 0 & 1 & -1 & 1 & 0 \end{bmatrix}$.

Player #1's next move, upon realizing that, once again, moving on a corner is more favourable than moving on an edge, will either be A or I (which are equivalent moves) or G. Here, a dilemma presents itself: Moving on G, the opposite corner that Player #2 chose previously, three 0's will become 1's, resulting in the matrix $\begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix}$. Moving on either A or I (suppose it is I), one -1 will become 0, one 0 will become 1, and one 1 will become 2, resulting in the matrix $\begin{bmatrix} -1 & 1 & 1 & 0 & 1 & 0 & 2 & 0 \end{bmatrix}$. It is not immediately apparent which resulting game matrix is more favourable for Player #1.

To answer this, study this particular sequence of moves:



The game matrix at the end of this sequence is $\begin{bmatrix} 1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$. Notice that, on Player #1's next turn (it is now Player #2's turn), they will have two opportunities to *fork* (that is, to threaten a win on more than one win position). If Player #1 plays A, they can win on R1 or C1; if I is chosen instead, they can win on R3 or C3. Notice that both moves will create two 2's in the game matrix, guaranteeing a win due to exclusion – Player #2 must be able to avoid both forks in order to stay alive, and simply playing a move on *one* of the spots where Player #1 could fork is not enough.

The only way for Player #2 to avoid a loss is to create a -2 in the game matrix, forcing Player #1 to choose a move that blocks the subsequent win. Of course, the move that Player #1 is forced to play cannot be the same move that produces the fork. Keeping this judgment in mind, any of the edge moves are perfectly adequate (and equivalent) moves in this scenario, as none of Player #1's responding moves will create two 2's in the game matrix, and all will produce a sequence of forced blocks that result in a draw.

Back to the game board at hand, we must consider subsequent moves in order to determine which choice is best. Let us dissect first the case where Player #1 moves on G. Player #2 must then respond ideally to the matrix $\begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix}$. Notice that, if nothing is done to counter it, Player #1 will have multiple ways to fork on their next turn

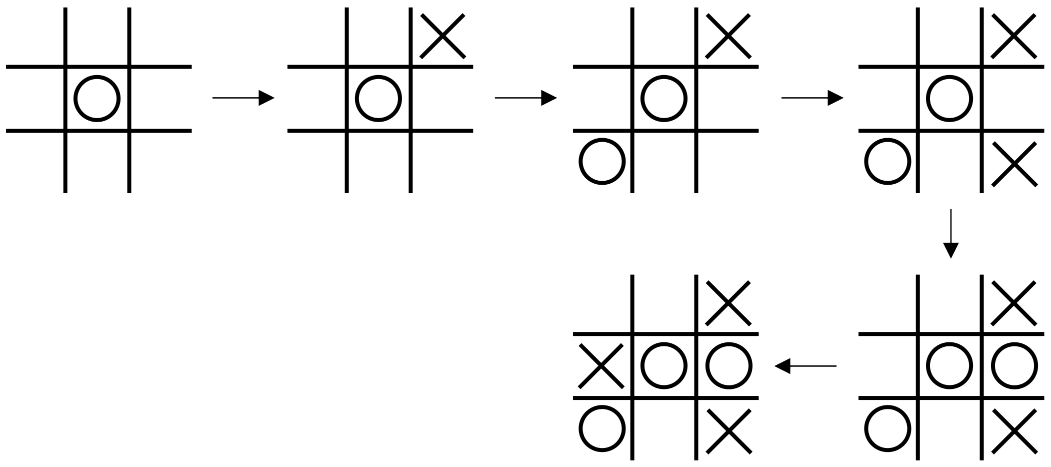
– all of A, D, H, and I will produce two 2's in the resulting matrix, securing the win for Player #1. Player #2 must select a move that forces Player #1 from selecting *any* of these moves, so the immediate candidates are A, B, F, and I (all create a -2 in the game matrix which must be blocked). Of these, B and F are not ideal, as playing them would still allow Player #1 to fork on their turn. A and I are thus the better choices, and the two are equivalent moves. Suppose I is chosen, so the game matrix becomes $\begin{bmatrix}-1 & 1 & 0 & 1 & 1 & -2 & 0 & 1\end{bmatrix}$.

Player #1 is now forced to move on F, as it is the only move remaining with a 1 in the C3 position, preventing Player #2 from winning. Subsequently, the game matrix becomes $\begin{bmatrix}-1 & 2 & 0 & 1 & 1 & -1 & 0 & 1\end{bmatrix}$, and Player #2 is forced to move on D to prevent Player #1 from winning on R2. It is not difficult to see that this sequence of moves will result in a draw.

Now, if Player #1 had chosen to move on I (or A) instead of G, Player #2 must respond to the matrix $\begin{bmatrix}-1 & 1 & 1 & 0 & 1 & 0 & 2 & 0\end{bmatrix}$. Here, they *must* move on A in order to prevent a loss on D1, resulting in the matrix $\begin{bmatrix}-2 & 1 & 1 & -1 & 1 & 0 & 1 & 0\end{bmatrix}$. Player #1 must then move on B to prevent a loss on R1. This particular sequence of forced moves will, again, result in a draw.

Here, the choice of move makes no difference, as both players are perfect (and are able to force a draw). However, against a non-perfect opponent, it is much better to set up opportunities to fork, thus guaranteeing the win, rather than to play a move that will inevitably be blocked. Hence, G (the opposite corner) is arguably the better move in this scenario.

This complete match of tic-tac-toe can be illustrated as follows:



(at which point the match can be considered a tie).

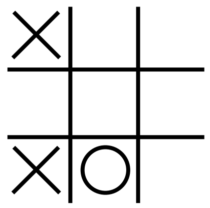
Considering many other examples like the one above, the following list of rules for Player #1 can be constructed and applied according to the current state of the game board. To acquire the same set of rules for Player #2, simply negate every number. In the case that more than one condition is satisfied, the guideline with the higher priority must be adopted.

1. Win – if there is a move that creates a 3 in the game matrix, play it.
2. Block – if there is a -2 in the game matrix, select a move that contains a 1 in that position.
3. Create Fork – if there is a move that creates two 2's in the game matrix, play it.
4. Block Fork – if the opponent has a move that creates two -2's from -1's, either create a 2 (as long as doing so will force them to choose a *different* move) or select a move that changes one of those -1's into a 0 (as long as there are not multiple forks).
5. Center – move on E.
6. Opposite Corner – if the opponent is on a corner and the opposite corner is vacant, play it.
7. Empty Corner – if a corner has not been selected yet, play it.
8. Empty Edge – if an edge has not been selected yet, play it.

This systematic approach to the game does, indeed, allow us to win or draw any match of tic-tac-toe from the beginning. If both players are perfect and utilize these guidelines, the match will inevitably end in a draw.

It is a simple matter to write a piece of code that is capable of processing a certain matrix, iterating through the conditions listed above, and selecting an appropriate move. Even more, the process of directly inputting the game matrix into the program (via some sort of loop, ideally), rather than translating some drawn-out board into numbers that can be interpreted by the computer, increases the speed at which an output can be received.

Since a program implementing this model is designed to play perfectly from the very beginning of the match, it may be interesting to input an in-progress game board that is "rigged" in a certain way and seeing how the program responds. Suppose the matrix $\begin{bmatrix} 1 & 0 & 0 & 2 & -1 & 0 & 1 & 1 \end{bmatrix}$ is fed into the program. Pictorially, the board is



Here, the computer is playing O as Player #2, and it has not played perfectly up to this point since it has moved on an edge rather than the center. Running through the above list of guidelines, the first that applies is to block the 2 in the C1 win position, and the only way this can be done is to move on D (with a 1 in the C1 position). However, Player #1 also has the opportunity to create a fork on their next move by moving on C or E, and, upon examination, the only way to avoid both of these is to move on E (threatening a win on C2). It is evident, then, that Player #2 has no way of forcing a tie, but our list of guidelines does allow Player #2 to *stay alive* as long as they can. This may not necessarily always be the case, and it may be a compelling problem to expand on our rules to accommodate an input where the player did not play perfectly and whose only objective is to continue the game as long as possible. We will leave this dilemma for future deliberation.

A minor shortcoming of this mathematical model of tic-tac-toe is that it does not completely remove the need to analyze several moves ahead in certain cases. In particular, in examining if the opponent has a move that creates a fork, blocking such a fork would require the program to search for the ideal move one move *ahead* of the current game state. This does decrease the efficiency of the program slightly, but a remedy for this drawback does not present itself readily. Perhaps this problem, again, could be dissected and resolved in future considerations of this model.

In summary, our rather simple representation of tic-tac-toe using matrices lends us immense insight into better move choices, overall strategy, and program efficiency considerations. For each game board, we were able to apply a set of guidelines constructed systematically using the model to determine the ideal move, and even expand a bit further on this notion by utilizing it on imperfect boards. If ever implemented into a piece of code, this model will do away with brute force, searching the game tree for the most favourable scenario (as what minimax was realized to do), and instead consider only the current game board (with the one exception mentioned above) and search for the best move in that one fixed scenario.

The following pseudocode is an example of the implementation of our ordered list of guidelines.

```
moves =
{
    A = [1  0  0  1  0  0  1  0]
    B = [1  0  0  0  1  0  0  0]
    C = [1  0  0  0  0  1  0  1]
    D = [0  1  0  1  0  0  0  0]
    E = [0  1  0  0  1  0  1  1]
    F = [0  1  0  0  0  1  0  0]
    G = [0  0  1  1  0  0  0  1]
    H = [0  0  1  0  1  0  0  0]
    I = [0  0  1  0  0  1  1  0]
}
RemainingMoves = {whatever moves haven't been chosen yet}
GameArray = [0  0  0  0  0  0  0  0]
```

```
define OppositeCorner (input)
{if input is A, output I; if input is C, output G; if input is G, output C,
if input is I, output A}
define NextMove =
{remove move from RemainingMoves, replace GameArray with GameArray + move}

check if 3 or -3 in GameArray
    if so:
        end
    if not:
        check if 3 can be created by adding move in RemainingMoves to
        GameArray
            if so:
                select move
                execute NextMove
        else check if -2 in GameArray in position i
            if so:
                select move from RemainingMoves with 1 in position i
                execute NextMove
        else check if two 2’s can be created by adding move in RemainingMoves
        to GameArray
            if so:
                select move
                execute NextMove
        else check if two -2’s can be created by subtracting move_fork in
        RemainingMoves from GameArray
            if so:
                check if 2 can be created by adding move in RemainingMoves to
                GameArray
                    if so:
                        check if two -2’s can be created by subtracting
                        move_next in RemainingMoves – move from GameArray +
                        move:
                            if not:
                                select move
                                execute NextMove
                    if not:
                        check if -2 in GameArray - move_fork in position i:
                            if so:
                                check if -2 in GameArray – move_fork + move
                                in RemainingMoves with 1 in position i
                                    if not:
                                        select move
                                        execute NextMove
        else check if E in RemainingMoves
            if so:
                select E
                execute NextMove
        else check if A, C, G, or I in RemainingMoves
            if so:
                for move in A, C, G, or I in RemainingMoves:
                    if OppositeCorner (move) in RemainingMoves:
                        select OppositeCorner (move)
                        execute NextMove
                else select Random from A, C, G, or I in RemainingMoves
                execute NextMove
        else
            select random from RemainingMoves
            execute NextMove

other player selects a move
execute NextMove
```

References

Fox, Jason. "Tic Tac Toe: Understanding The Minimax Algorithm." *Never Stop Building.* N.p., 23 Dec. 2013. Web. 25 Aug. 2017. <http://neverstopbuilding.com/minimax>.

"Tic-Tac-Toe Algorithm." Blog post. *Omnimaga.* N.p., 26 Mar. 2012. Web. 25 Aug. 2017. <https://www.omnimaga.org/math-and-science/tic-tac-toe-algorithm/>.

Crowley, Kevin, and Robert S. Siegler. "Flexible Strategy Use in Young Children's Tic-Tac-Toe." *Cognitive Science* 17 (1993): 531-61. *Wiley Online Library.* Web. 25 Aug. 2017. <http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1704_3/epdf>.