An Analysis of Heuristics for a Mathematically Incomplete Variant of TicTacToe

Feyaz Baker*, Arunava Mukhoti, B. R. Chandavarkar

Department of Computer, Science and Engineering, National Institute of Technology Karnataka, Surathkal, Mangalore, India

*Corresponding author
doi: https://doi.org/10.21467/proceedings.114.57

Abstract

We attempt to produce a game-winning heuristic for the mathematically incomplete game Ultimate Tic Tac Toe (UTT). There are several game AI that use Monte Carlo Tree Search to decide moves, however, heuristics offer a faster and computationally cheaper alternative. The mathematical analysis of UTT has not been actively pursued, so we attempt to prove a posteriori. We have decided on a few strategies for playing, and assign different strategies to each player. We play several automated games of UTT, and statistically analyse which games end quickest, and use that data to find optimal strategies for playing. This can be used to produce game heuristics for more complicated games, and produce insight about strategies. The first objective is to specify a framework that can compare heuristics for UTT, and decide an optimal strategy for both players. The second objective is to test the framework with a large amount of data, and produce demonstrable results for UTT. Lastly, to aid further research in this topic, we release our dataset into the public domain.

Keywords: Game Heuristics, Brute Force Heuristic Analysis, Ultimate Tic Tac Toe, Monte Carlo Tree Search

1 Introduction

Ultimate Tic Tac Toe (UTT) is a stronger version of the common Tic Tac Toe (TTT). TTT has simple rules, in a 3x3 grid, players take turns marking a cell, as either X or O. X goes first. The first player to complete three symbols in a line, either diagonal or row or column, wins the game.

UTT has slightly convoluted rules by comparison. It is played on a 3x3 grid, with each cell of the major 3x3 grid split into a smaller 3x3 grid. We call the entire resulting 9x9 grid as the game board/global grid, and call each of the smaller grids as boxes/local grids. The goal is to win games at the smaller level, i.e., play games inside a single box to completion. Once a box has been claimed by either player, the corresponding cell in the global grid is marked in that player's symbol. The game ends when a player has successfully captured three cells in a line, on the global grid.

There's one peculiarity of the game we will now discuss. This rule is the reason the game has eluded mathematical analysis. There's a weaker version of UTT that employs a weaker version of the rule, and that version has been proven mathematically complete.

The rule is, if a player makes a move in any cell in a local grid, their opponent will have to make their move in the corresponding box of the global grid (See Fig 2). If the opponent is sent to a box that is already claimed, they can choose any box, and then make their move in that box. At the start of the game, X has to similarly first pick a box, and then make a move in that box.

Another factor that adds complexity is the case of a draw in any local grid. This would reflect as a null value in the global grid, and that cell in the global grid can be used by either party to produce three symbols in a line (Fig 3). It is even possible to end a game with no winner, if the cells on global grid are claimed with

Proceedings DOI: 10.21467/proceedings.114; Series: AIJR Proceedings; ISSN: 2582-3922; ISBN: 978-81-947843-8-8

no clear winner. Game theory tries to process move-based games by developing a theory based on the quantifiable outcomes of every move. When all such rules of game theory are put together into a function, we can feed in a game state and find an optimal move. Such an automaton would require lengthy analyses, and a deep understanding of the game. The nuances of UTT complicate things further. This motivates us to develop a heuristic which can improve our chances of victory.

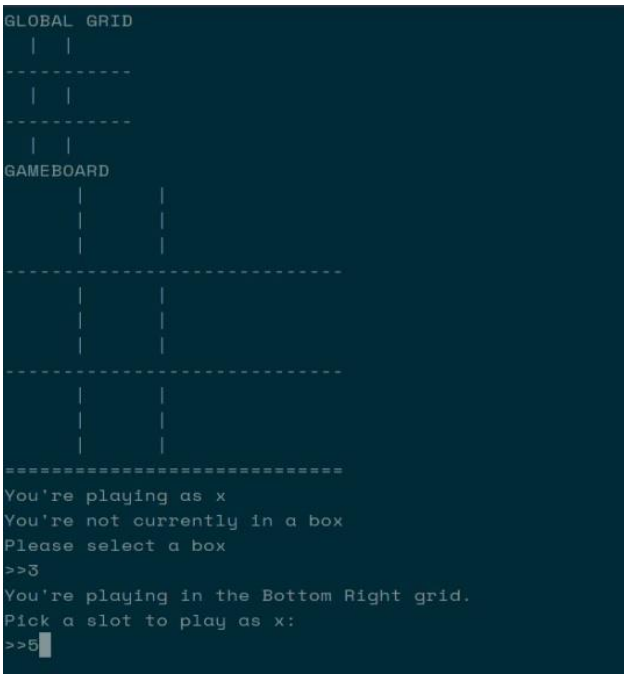

Fig. 1. X makes a move in the middle middle cell of the bottom right grid.

Fig. 2. This move sends O to the middle middle box.

The layout of the paper is as follows: Section 2 discusses the results of current literature, notably [1], which follows the slightly weaker version of UTT and presents a winning game strategy. Section 3 lists our methodology, and explains the foundational assumptions of our paper. Section 4 presents an outline for our

simulation, lists the strategies we have considered for UTT, and gives some basic terminology. Pseudocode is also listed to better analyse the strategies. Section 5 summarises the results of our paper, along with three tables outlining all the data produced. Section 6 presents possible avenues of research for future work in this field.

2 Literature Review

Several weaker versions of UTT have been proposed, that have produced verifiably optimal strategies [1]. In Berthelon et. al. [1], it was assumed that even after a box was claimed, it would be possible to make moves in it. This takes away the complexity of optimally claiming choice pieces, like corners or edges, before taking the centre piece.

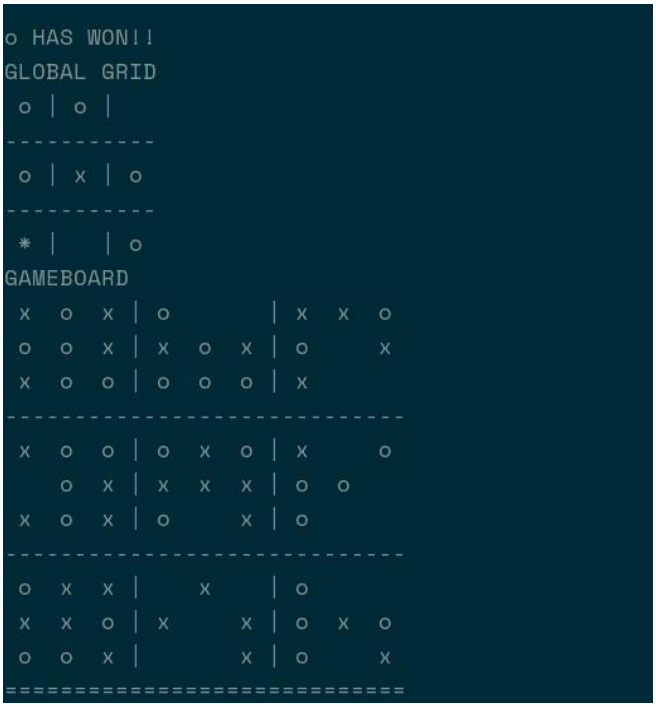

Fig. 3. A game won with a draw in a local grid

Algorithms like the Monte Carlo Tree search have given high degree of accuracy based on some assumptions. There have been several working playable implementations of Ultimate Tic Tac Toe and some of them use Monte Carlo Tree Search [2]. By assigning some game states as more valuable than others, it is possible to produce a tree structure reaching out from the current game state, and choose the branches that produce the highest value. These approaches invariably take high computational power, and get slower with the depth of search. However, the foundational assumption, that some particular states are more favourable than others, have not been successfully proved outside of analogy to the standard game of Tic-Tac-Toe. In this work, we attempt to verify if the simplifying assumptions in most tree search implementations are accurate, ie, are some pieces more arbitrarily more valuable than others, or is there a higher dependency on the game state than previously assumed.

From the Hand et. al [3], we understand that a relatively simple classifier that assumes independence of input variables can sometimes outperform other complex rule based classifiers. We extend this theory to state that, by measuring the win rate and average game length of a large sample of games, where each player follows a different strategy, we can glean the relative success rates of a strategy against others. This information can be used to devise an optimal strategy for UTT, and even improve assumptions in tree based search algorithms.

Proceedings DOI: 10.21467/proceedings.114
ISBN: 978-81-947843-8-8

Series: AIJR Proceedings
ISSN: 2582-3922

3 Hypothesis and Experimentation

We first made an implementation of the game in C++. We assume that there may be different strategies that produce a higher rate of victory. Since we do not have an efficient algorithm to take a game state and predict who is in a more advantageous position, we can make an arbitrary a priori assumption. If there exists an optimal game strategy, and both players use it, it would result in the game finishing sooner than a series of non-optimal moves on the parts of both players.

For reference, we present the pseudo-code of an automated game. The entire code is available on our repo here.

4 Implementation Details

For completeness, we present a short pseudocode of the logic used in the fully automated version of UTT. To help the reader gain a clearer understanding of the below code, we have used certain terminology based on the layout of a numeric keypad.

- Key 1 - Bottom-left cell
- Key 2 - Bottom edge cell
- Key 3 - Bottom-right cell
- Key 4 - Left edge
- Key 5 - Center cell
- Key 6 - Right edge cell
- Key 7 - Top-left cell
- Key 8 - Top edge cell
- Key 9 - Top-right cell

4.1 PSEUDOCODE:

```
Initialise game boards (3x3 global board
and 9x9 local grids).

def move(input):
  // Simulates one move in a game, for input
    if(current box is unspecified or invalid):
      take input as the new box to play in.
    else if(input cell is already filled):
      request a different input.
    else
      make a move and hand over to next player.
      set box of next player
      corresponding to move made.
def checkwinner():
  // Returns the winner for a given game board.
  for each box of local grid
      find winner of box
    if(winner exists)
      mark winner in global grid.

  if(global grid has null cells)
    convert all null cells to favourable state.
  if(global grid has a winner)
```

```
    winner = current player
  else
    convert all null cells to unfavourable state.
    if(global grid has a winner)
      winner = last player
    else
      winner = None
  return winner

while(global grid is not filled)
  move()
  winner = checkwinner()
  if winner!=empty:
    print('Game ended, ' ,winner 'won')
```

With this implementation in pseudocode, we are free to produce move-making strategies. We assume, from our move() method, that the optimal choice of box is not different from the optimal choice of cell. We also assume that rand() will generate a random number between 0 and INT MAX as defined in C++.

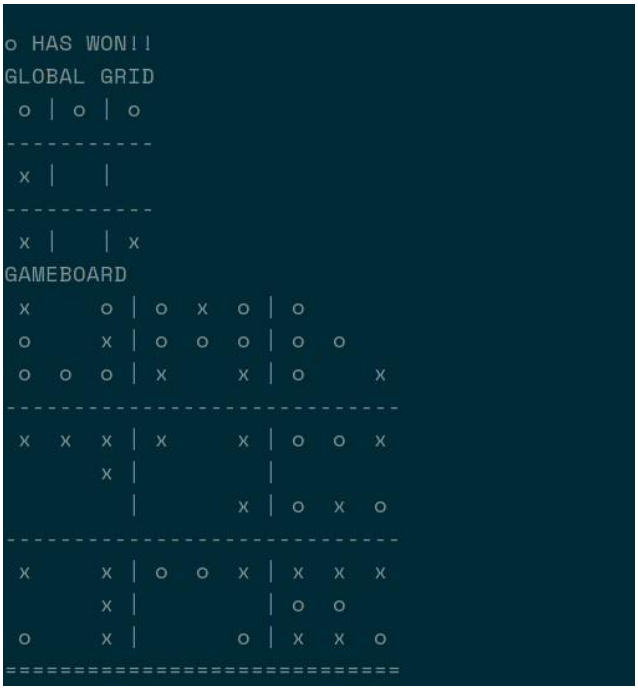

Fig. 4. A game simulated with priority given to corners. It is very visible where most moves are made, and correspondingly, more boxes are claimed.

A completely random move selection would not be efficient in testing specific theories, so we decided to assign weights for each piece, and use the weights to give priority to different pieces. For example, if we decide a corner piece is worth more than an edge piece, we assign the weight of an edge piece as x, and assign weight of a single corner piece as kx, where k can be any constant. In this way, we present the following heuristic strategies to compare:

Strategy 0: Randomly choose any of the 9 cells. This strategy is a representation of the random, unplanned moves of a beginner. The game state is ignored, and this strategy serves as a baseline to measure other strategies.

```
def strategy0():
  return rand()%9+1
```

Strategy 1: Prioritizing the corners over the edges, and the edges over the center.

```
def strategy1():
  randomnum = rand()%84

  if (randomnum<= 15)
    choose bottom-left cell
  else if (randomnum <= 31)
    choose bottom-right cell
  else if (randomnum <= 47)
    choose top-left cell
  else if (randomnum <= 63)
    choose top-right cell
  else if (randomnum <= 67)
    choose bottom edge cell
  else if (randomnum <= 71)
    choose left edge cell
  else if (randomnum <= 75)
    choose right edge cell
  else if (randomnum <= 79)
    choose top edge cell
  else if (randomnum <= 83)
    choose center cell
```

Strategy 2: Prioritize the corners over the center, and the center over the edges.

```
def strategy2():
  randomnum = rand()%21

  if (randomnum <= 15)
    choose bottom-left cell
  else if (randomnum <= 31)
    choose bottom-right cell
  else if (randomnum <= 47)
    choose top-left cell
  else if (randomnum <= 63)
    choose top-right cell
  else if (randomnum <= 79)
    choose center cell
  else if (randomnum == 80)
    choose bottom edge cell
  else if (randomnum == 81)
    choose left edge cell
  else if (randomnum == 82)
    choose right edge cell
  else if (randomnum == 83)
    choose top edge cell
```

Strategy 3: Prioritizing the center over the edges, and the edges over the corners.

```
def strategy3():
  randomnum = rand() %84

  if (randomnum <= 63)
    choose center cell
```

```
  else if (randomnum <= 67)
    choose bottom edge cell
  else if (randomnum <= 71)
    choose left edge cell
  else if (randomnum <= 75)
    choose right edge cell
  else if (randomnum <= 79)
    choose top edge cell
  else if (randomnum == 80)
    choose bottom-left cell
  else if (randomnum == 81)
    choose bottom-right cell
  else if (randomnum == 82)
    choose top-left cell
  else if (randomnum == 83)
    choose top-right cell
```

Strategy 4: Prioritizing the center over the corners, and the corners over the edges.

```
def strategy4():
  randomnum = rand() %84

  if (randomnum <= 63)
    choose center cell
  else if (randomnum <= 67)
    choose bottom-left cell
  else if (randomnum <= 71)
    choose bottom-right cell
  else if (randomnum <= 75)
    choose top-left cell
  else if (randomnum <= 79)
    choose top-right cell
  else if (randomnum == 80)
    choose bottom edge cell
  else if (randomnum == 81)
    choose left edge cell
  else if (randomnum == 82)
    choose right edge cell
  else if (randomnum == 83)
    choose top edge cell
```

Strategy 5: Prioritizing the edges over the center, and the center over the corners.

```
def strategy5():
  randomnum = rand()%84

  if (randomnum <= 15)
    choose bottom edge cell
  else if (randomnum <= 31)
    choose left edge cell
  else if (randomnum <= 47)
    choose right edge cell
  else if (randomnum <= 63)
    choose top edge cell
  else if (randomnum <= 79)
    choose center cell
  else if (randomnum == 80)
```

Proceedings DOI: 10.21467/proceedings.114
ISBN: 978-81-947843-8-8

Series: AIJR Proceedings
ISSN: 2582-3922

```
    choose bottom-left cell
  else if (randomnum == 81)
    choose bottom-right cell
  else if (randomnum == 82)
    choose top-left cell
  else if (randomnum == 83)
    choose top-right cell
```

Strategy 6: Prioritizing the edges over the corners, and the corners over the center.

```
def strategy6():
  randomnum = rand()%84

  if (randomnum <= 15)
    choose bottom edge cell
  else if (randomnum <= 31)
    choose left edge cell
  else if (randomnum <= 47)
    choose right edge cell
  else if (randomnum <= 63)
    choose top edge cell
  else if (randomnum <= 67)
    choose bottom-left cell
  else if (randomnum <= 71)
    choose bottom-right cell
  else if (randomnum <= 75)
    choose top-left  cell
  else if (randomnum <= 79)
    choose top-right cell
  else if (randomnum <= 83)
    choose center cell
```

Strategy 7: Random corner fixation. This strategy resembles the optimal game strategy for Tic-Tac-Toe. The strategy emphasises picking a specific corner first and then selecting its adjacent corners 5. At this point, it forms a very distinct triangle pattern, which is a winning state no matter what move the opponent makes. The only counter for this is if the opponent also claims corners exclusively, leading to each player having two corners.

```
def strategy7():
  For each grid, assign a corner to X and O. This corner
  will be the first priority in capturing, for that player,
  in that grid.
  //This description assumes corner is in top right.
  randomnum = rand()%25

  if (randomnum<=1):
    choose center cell
  if(randomnum<=3):
    choose top edge cell
  if(randomnum<=5):
    choose right edge cell
  if(randomnum<=9):
    choose top left cell
  if(randomnum<=13):
    choose bottom right cell
```

```
if(randomnum<=21):
   choose top right cell
if(randomnum==22):
   choose left edge cell
if(randomnum==23):
   choose bottom edge cell
if (randomnum==24):
   choose bottom left cell
```

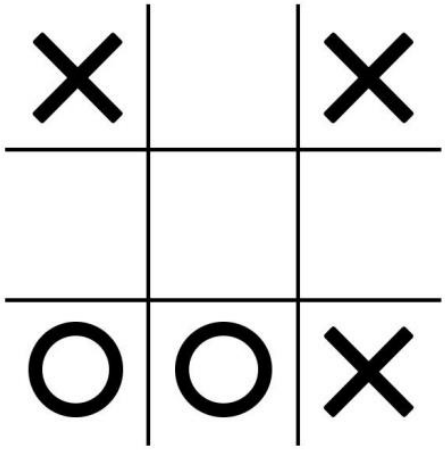

Fig. 5. An example of the three corner priority in Tic Tac Toe, with the top right corner as highest priority.

We have implemented strategy 7 for UTT to reflect this simple and efficient algorithm. Each player, X and O, is allotted a corner that they prioritize for each grid. This allocation is done randomly, and under the assumption, the choice of corners wouldn't have an optimal sub-strategy. After this allocation, every time a player is sent to a grid, the optimal corner is retrieved, and the move is made accordingly. We've made four helper functions, one for each corner, and assigned a switch case to route the results as needed.

We can easily prove that following any of the above strategies produces a noticeable change in the board's final state. The above Fig 4 was generated by giving the highest priority to corner pieces and least preference to the centrepiece. We see the board's imbalanced state and how prioritizing corner pieces leads to more of the corners being claimed than edge pieces, vice versa for edges and centrepieces.

5 Results

Once we have a significant sample size for our data, we can extrapolate using a simple Naive-Bayes classifier and argue that some strategies are more representative of winning games than losing games. With a significant number of samples, we can even make an AI play against someone without any human intervention. In the spirit of open research, we will be making the dataset public after this paper. We have been working with eight different strategies. To test these, we have generated game data for 1000 games for every combination of these eight strategies. We compare the results of a total of 64,000 rounds and observe the results. Strategy 0 was placed to decide if playing at random was gave any significant deviation from the norm. Strategies 1 through 6 are the six permutations for weights by classes, i.e., where we treat each class of pieces (corner pieces, edge pieces, centrepiece) as a different class, and thus a different priority of capture. Strategy 7 measures the accuracy and efficacy of using an optimal method for normal TTT, adapted to UTT. We present our results in Tables I, II and III, where the row indices correspond to the strategy employed by X, and the column indices correspond to the strategy employed by O. Table I shows the percentage of wins

Proceedings DOI: 10.21467/proceedings.114
ISBN: 978-81-947843-8-8

Series: AIJR Proceedings
ISSN: 2582-3922

by X and O, Table II shows the average number of moves made in games won by X, and O respectively, while Table III shows the percentage of games ending in a draw, and the average number of moves made in those games.

We observe that when both players play randomly, we end up with a more significant number of moves till completion, on average. We notice that X's highest win rate is at 75.8%, when X was playing strategy 4 against O on strategy 6. We see that when both players follow the same strategy, there is a considerably more significant percentage of drawn games. There is also a bias towards X in those situations. Presumably, X can win simply because it has made more moves than O has. O has won the most number of times when X was playing strategies 5 and 6, meaning capturing edges is a comparatively weaker move for X.

In almost all the games we simulate, X has won the most times. This starting advantage reverses when both players prioritize capturing edge pieces. This means that there is a high probability that playing edge pieces is an optimal strategy for O but may not always be so for X.

The case that produces the most wins for O is when playing strategy 4 against X with strategy 5, at a 69.6% success rate. O's second-highest wins are when O plays strategy 7 against X's 6, at 69.1%. O has less than 50% victory rate in majority cases, barring cases when X plays either strategies 5, and 6. To verify correctness of the data, we ran the experiment several times and always produced less than 5% deviation from victory rates.

The length of a game that ends in a draw is always significantly more than any that ends in a victory. This backs up our assumption that any game-winning strategy would finish in the least number of moves, rather than spend time lining up grids before capturing. We see that the average number of moves it takes for X is around the same for O, which seems to indicate that any optimal method would work for X as well as O. The largest number for moves till victory appears when both players follow strategy 6.

Ordinary TTT prioritizes according to strategy 7, and playing with two corners is the norm for an optimal game. This strategy does not seem to apply for UTT, or if there is an advantage, it cannot be demonstrated by this experiment.

The most draws are when X plays by strategy 6 and O by strategy 1. It seems like those strategies cancel out the optimal moves for each other and lead to more games that are unwinnable as a result.

6 Conclusion And Future Work

By probabilistic analysis, it seems that X has an undue advantage due to making the first move. For X to win, the choice of strategy is always dependent on O, but O's victory always seems dependent on prioritizing edges over other systems. As X, it is always safest to play corners aggressively. As O, avoiding corners will prove some benefit. We verify that the direct translation of strategy 7 into UTT is not a winning metric. Heuristic measures used in Tree Search algorithms can use this data to develop better pruning and searching methods. As a shorthand rule, it is far easier to win games of UTT as X than as O.

TABLE I: Percentage of wins by X and O.

X	0	1	2	3	4	5	6	7
0	52.4% , 47.6%	54.9% , 45.1%	53.3% , 46.7%	53.3% , 46.7%	46.6% , 53.4%	59.3% , 40.7%	64.4% , 35.6%	47.8% , 52.2%
1	49.2% , 50.8%	52.5% , 47.5%	54.1% , 45.9%	45.4% , 54.6%	44.5% , 55.5%	61.6% , 38.4%	60.8% , 39.2%	47.5% , 52.5%
2	53.5% , 46.5%	55.6% , 44.4%	54.6% , 45.4%	54.1% , 45.9%	52.7% , 47.3%	68.0% , 32.0%	63.8% , 36.2%	46.1% , 53.9%
3	51.3% , 48.7%	58.6% , 41.4%	52.7% , 47.3%	55.9% , 44.1%	45.9% , 54.1%	55.0% , 45.0%	62.6% , 37.4%	45.2% , 54.8%
4	58.6% , 41.4%	57.8% , 42.2%	53.2% , 46.8%	68.1% , 31.9%	55.1% , 44.9%	74.8% , 25.2%	75.8% , 24.2%	52.2% , 47.8%
5	44.4% , 55.6%	41.4% , 58.6%	37.0% , 63.0%	47.6% , 52.4%	30.4% , 69.6%	56.0% , 44.0%	57.8% , 42.2%	33.9% , 66.1%
6	36.7% , 63.3%	41.2% , 58.8%	36.0% , 64.0%	44.6% , 55.4%	27.8% , 72.2%	45.6% , 54.4%	50.6% , 49.4%	30.9% , 69.1%
7	59.7% , 40.3%	58.9% , 41.1%	58.5% , 41.5%	58.6% , 41.4%	52.5% , 47.5%	73.5% , 26.5%	68.0% , 32.0%	54.9% , 45.1%

TABLE II: Average number of moves in games won by X, and O respectively

X	0	1	2	3	4	5	6	7
0	56.1 ,56.6	52.9 ,53.6	46.6 ,49.4	48.8 ,50.5	48.5 ,50.5	52.6 ,54.1	56.7 ,57.6	53.2 ,53.3
1	53.6 ,53.0	52.2 ,52.7	44.4 ,47.8	46.6 ,47.0	45.0 ,48.4	49.4 ,51.2	54.0 ,54.1	52.0 ,51.7
2	49.1 ,46.3	47.7 ,44.1	44.0 ,44.0	45.8 ,44.9	45.3 ,46.9	46.5 ,45.6	49.6 ,49.1	47.1 ,43.8
3	51.5 ,48.6	46.9 ,46.6	45.5 ,45.2	51.2 ,52.2	46.8 ,46.8	50.2 ,46.9	55.6 ,51.1	46.5 ,45.8
4	50.2 ,48.3	49.1 ,45.9	48.3 ,45.3	47.3 ,48.8	46.4 ,47.3	45.8 ,48.0	52.5 ,51.9	48.9 ,45.1
5	54.4 ,52.9	51.0 ,49.7	45.8 ,46.9	48.3 ,50.9	48.0 ,47.0	53.7 ,54.7	55.0 ,55.3	51.0 ,48.8
6	57.3 ,56.9	54.3 ,54.1	49.1 ,50.0	51.1 ,55.7	51.4 ,52.2	54.7 ,55.0	58.7 ,59.3	54.1 ,54.1
7	53.0 ,53.1	51.3 ,51.8	43.8 ,47.4	46.2 ,46.8	44.7 ,47.7	48.0 ,51.4	53.5 ,53.7	51.1 ,51.1

TABLE III: Percentage of games ending in draw, and average number of moves in those games.

X	0	1	2	3	4	5	6	7
0	21.9%, 60.9	16.2%, 58.7	8.7%, 54.7	16.3%, 56.7	15.5%, 55.5	13.3%, 58.9	27.2%, 61.7	14.9%, 58.5
1	15.4%, 58.4	16.2% , 58.9	10.3% , 54.5	12.7% , 53.7	13.8% , 55.7	8.3% , 56.5	37.2% , 59.4	15.5% , 58.2
2	9.8% , 55.1	10.5% , 55.2	12.3% , 54.8	8.3% , 53.1	14.2% , 55.3	5.0% , 54.1	17.1% , 54.7	9.9% , 54.9
3	17.5% , 57.3	11.8%, 53.0	5.9% , 52.6	24.5%, 60.2	12.9% , 54.8	21.3% , 57.4	25.7% , 60.8	13.4% , 53.4
4	13.1% , 56.4	12.3% , 55.7	13.8% , 55.7	16.0% , 55.0	14.1% , 56.3	7.6% , 55.2	24.9% , 57.0	12.5% , 55.0
5	16.3% , 58.8	8.3% , 56.3	5.5% , 53.7	20.8% , 57.5	7.0% , 56.0	16.2% , 59.6	25.3% , 60.9	9.8% , 55.5
6	30.7% , 61.7	36.7% , 59.7	20.8% , 54.2	24.6% , 60.2	29.6% , 56.1	25.8% , 60.4	23.3% , 63.9	28.5% , 59.4
7	16.6% , 58.6	14.6% , 58.2	9.5% , 53.9	12.5% , 53.2	12.4% , 54.4	9.1% , 56.1	28.8% , 58.9	12.9% , 57.5

Proceedings DOI: 10.21467/proceedings.114
ISBN: 978-81-947843-8-8

Series: AIJR Proceedings
ISSN: 2582-3922

In this paper, a framework capable of proving efficacy of heuristic strategies for turn based games was presented, with a proof stemming from Naive Bayes, namely, that if a strategy has produced significant advantages in many situations, it is likely to provide that advantage in other situations. This framework can be expanded to test methods for different games, and serves as a useful tool to verify basic assumptions and heuristics.

To address RNG move selection, it bears mentioning that the C++ implementation has a modulo bias. Any higher dimension implementation of the game will have to adjust accordingly. This work is only intended to be the first step into this field. Game theory has not yet used brute force data analysis to justify its theories. To this end, we have provided all our simulated game data, of roughly 64,000 game samples, on our Git repository. We have also assumed that the choice of box is no different from the choice of a move to make. The results we obtain will be biased by this assumption.

References

[1] G. Bertholon, R. Geraud-Stewart, A. Kugelmann, T. Lenoir, D. Nac-´ cache, At most 43 moves, at least 29: Optimal strategies and bounds for ultimate tic-tac-toe, arXiv preprint arXiv:2006.02353 (2020).

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484–503. URL http://www.nature.com/nature/journal/v529/n7587/full/ nature16961.html

[3] D. J. Hand, K. Yu, Idiot's bayes —not so stupid after all?, International Statistical Review 69 (3) (2001) 385–398.