Proceedings of the 2009 IEEE
International Conference on Information and Automation
June 22 -25, 2009, Zhuhai/Macau, China

Implementing a No-Loss State in the Game of Tic-Tac-Toe using a Customized Decision Tree Algorithm

Sivaraman Sriram, Rajkumar Vijayarangan, Saaisree Raghuraman,
Xiaobu Yuan[*], Senior Member, IEEE.
School of Computer Science, University of Windsor, Windsor, ON- N9B3P4
{sriram,vijayarr, raghuras, xyuan}@uwindsor.ca

Abstract: **- The game of Tic-Tac-Toe is one of the most commonly known games. This game doesn't allow one to win all the time and a significant proportion of games played results in a draw. This study is aimed at evolving of no-loss strategies in the game using Decision Tree Algorithm and comparing them with existing methodologies, mainly focused on the implementation of the game using the Minimax algorithm. The Minimax algorithm does provide an optimal No-Loss Strategy by assuming that both players play optimally. So the question that comes out is what happens when the opponent plays un-optimally, in these cases the minimax proved to play non optimal moves, even though it wins at the next state rather than the expected state. Thus this paper provides a clear study of those trivial states and provides an optimal game play using an decision tree algorithm independent of the opponent's game strategy.**

I. INTRODUCTION

The games such as Tic Tac Toe, Checkers, Chess etc are known as 'Zero sum' games. All these games have at least one thing in common; they are logic games [1]. They are also known as 'Full Information' games. Each player knows everything about the possible moves of the adversary. These games are known as 'Zero sum' games because the gains of one player are equal to the losses of another player where as in 'Non-Zero sum' games both players can suffer gains or losses together. No-Loss Strategy is a strategy in which the terminal state for the game of Tic Tac Toe would be either a win or a draw. The main objective of this paper is to build no-loss strategies in the game using Decision Tree Algorithm and comparing them with existing methodologies, such as Minimax algorithm. We were successful in analyzing the differences, advantages, disadvantages of both algorithms. Firstly, the implemention was done using Minimax algorithm and used it as a work bench for the comparison. Secondly, we implemented Tic Tac Toe using Decision tree. Finally, some experimentation was done so as to analyze the performance of these algorithms.
A Decision tree is a top–down greedy search classifier in the form of a tree structure where each node is either a leaf node or a decision node. A leaf node is one which indicates the value of the target attribute. A decision node is one which specifies some test to be carried out on a single attribute value with one branch and sub-tree for each possible outcome of the test. A Minimax search is a recursive algorithm for finding the next move of a given player. It determines all possible continuations of the game to the desired level evaluating each possible set of moves and assigning a score [2]. The search then steps back-up through the game tree and alternates between choosing the highest child score at nodes representing the first player and the lowest child score at nodes representing the second player. Game tree search is aimed at finding optimal strategy for the game.

II. PREVIOUS RELATED WORK

A. Tic-Tac-Toe Game

Tic-Tac-Toe is an interesting two player game in which each player takes a turn by marking the spaces in a 3x3 grid. Normally, these spaces are marked by the symbols 'X' and '0' where X represents player one and 0 represents player 2 or vice versa. Tic-Tac-Toe produce results of three types which can be a "win", "loss" or a "draw". The player who succeeds placing three respective marks in a horizontal, vertical or diagonal row/column wins the game. There is both feasible and infeasible game states associated with this game.
There are certain rules by means of which we determine these states. Those conditions have been explicitly explained in the paper which are as follows [1]:

(a) The number of 'X' in any given state would be one more than or equal to number of '0' where we assume X is played first.
(b) Both players would not move if the other has won.

These conditions play a vital part in determining the feasible game states and thereby eliminating the infeasible game states.

X	X	X
X	X	O
O	X	X

X	X	X
O	O	O

X	X	X
O		O
O		

Figure 1: Infeasible Game States [1]

B. Prior works

We would like to highlight some of the major contributions made by researchers in evolving "No Loss Strategies" for the game of Tic Tac Toe. The earlier work done by the people Hochmuth [3] and Soedarmadji [4] seems to be not clear and effective. Soedarmadji suggested a decentralized decision making to find a competent strategy which forces a draw or a win depending on the proficiency of the opponent player. The paper[1], provides a clear evidence that Soedarmadji's[4] No Loss Strategy fails in three different scenarios and evolves a number of No-Loss Strategies in comparison with existing methodologies. The three different scenarios are explained by means of the diagram shown below:

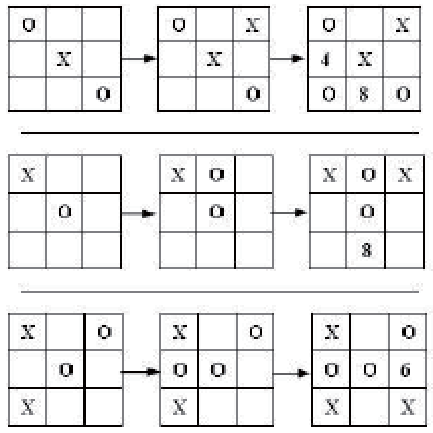


Figure 2: Three cases in which Soedarmadji's Heuristic solution loses the game. X is played by heuristic solution [1]

The work done by Hochmuth[3] reported a single No Loss Strategy but the study failed to explain the strategy in detail. The efficiency of parallel Minimax algorithm is discussed in the paper [2]. This paper brings out the pros and cons of Minimax algorithm and also explains an optimization strategy for the same using Alpha, Beta Cutoff. Performance comparison has been made for hybrid (multilevel,) flat and multithreaded parallel programming models. Speedup and efficiency as well as scalability in respect to size of the multi-computer and its impact on the performance of the parallel system have been estimated on the basis of experimental results. The communication/ computation ratio (CCR) of the parallel hybrid and flat implementations of the Minimax algorithm has been estimated. We also took a clear look of the *-Minimax performance with respect the game 'Backgammon' by Thomas Hauk[5]. The paper provides the first performance results for Ballard's*-Minimax algorithms applied to real-world domain. The paper also presents empirical evidence that with today's sophisticated evaluation functions; good checker play in backgammon does not require deep searches. The resource taken from North Western University [6] explains the decision tree in detail.[6] describes decision tree with the game 5-coin puzzle and explicitly shows the game trees associated. The diagram below explains the decision tree for the game of 5-coin puzzle:

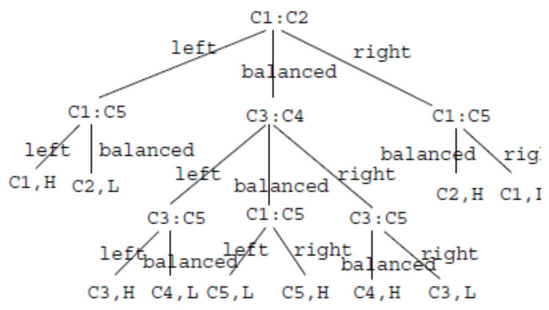


Figure 3: Decision tree for the 5-coin puzzle[7]

III. PROPOSED WORK

In this section, we would like to describe the methodologies we used for developing and comparing the search algorithms (Minimax and Decision tree) specific to the game Tic Tac Toe.

A. Minimax in Tic Tac Toe

The Minimax algorithm is a recursive algorithm using which we can compute the possible moves made by the players in the near future. There are two players involved, MAX and MIN. A search tree is generated using Depth First Search starting with a current game position up to all possible end game positions. Then the final game position is evaluated from MAX's point of view. Afterwards, the inner node values of the tree are filled Bottom-up with the evaluated values. The nodes that belong to the MAX player will receive the maximum value of its children. The nodes for the MIN player will select the minimum value of its children. The MAX player will try to select the move with highest value in the end. But the MIN player also has something to say about it and he will try to select the moves that are better to him, thus minimizing MAX's outcome.

The Minimax algorithm consists of five steps [7]:

Step 1: Generate the whole game tree, all the way down to the terminal states

Step 2: Apply the utility function to each terminal state to get it's value.

Step 3: Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree.

Step 4: Continue backing up the values from the leaf nodes towards the root, one layer at a time.

Step 5: Eventually, the backed up values reach the top of the tree; at that point, MAX chooses the move that leads to the highest value.

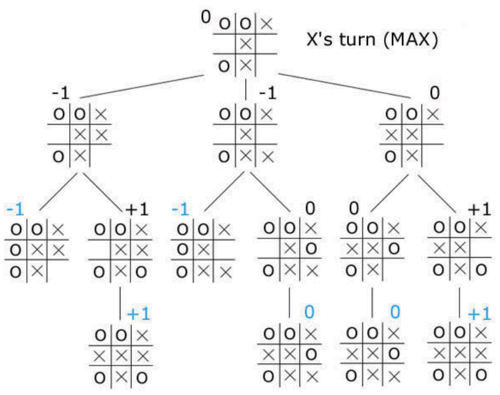


Figure 4: Partial Search Tree for Tic Tac Toe[8]

Above is a section of a game tree for Tic Tac Toe. Each node represents a board position, and the children of each node are the legal moves from that position. To score each position, we will give each position which is favorable for player 1 a positive number (the more positive, the more favorable). Similarly, we will give each position which is favorable for player 2 a negative number (the more negative, the more favorable). Here, player 1 is 'X', player 2 is 'O', and the only three scores we will have are +1 for a win by 'X', -1 for a win by 'O', and 0 for a draw. Note here that the blue scores are the only ones that can be computed by looking at the current position. To calculate the scores for the other positions, we must look ahead a few moves. Instead of knowing the full path that leads to victory, the decisions are made with the path that might lead to victory. If the optimization isn't well chosen, or it is badly applied, then we could end up with a dumb AI. And it would have been better to use random moves. Heuristics are knowledge that we have about the game, and it can help generate better evaluation functions. One of the reasons that the evaluation function must be able to evaluate game positions for both players is that you don't know to which player the limit depth belongs. However having two functions can be avoided if the game is symmetric. This means that the loss of a player equals the gains of the other. Such games are also known as 'Zero sum' games. For these games one evaluation function is enough, one of the players just have to negate the return of the function.

B. Decision tree in Tic Tac Toe

A decision tree represents a hierarchy of several courses of action. The tree starts with a decision that has to be made and branches out to subsequent (or possible) ones lead by the outcome of initial decision. Thus, from the above mentioned logic we would like to incorporate the same for the game of Tic Tac Toe. A decision tree for Tic Tac Toe can be drawn and can be analyzed for different No-Loss Strategies. The decision tree is split up into two different parts; one for the first move made by the opponent and second is generalized for all other moves. The decision trees for Tic Tac Toe are as follows:

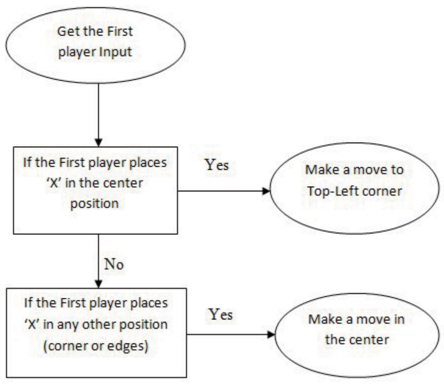

Get the First player Input
If the First player places 'X' in the center position
Yes
Make a move to Top-Left corner
No
If the First player places 'X' in any other position (corner or edges)
Yes
Make a move in the center

Figure 5: Decision tree for the first move

From the tree constructed, the algorithm gets input from first player(Human) that is 'X' and finds out where the player places 'X' in the nine different spaces. It checks for two conditions which are, if the first player makes the move in the center space, then the algorithm is fixed to make a move '0' in any of the four corner spaces which is at top-left corner or if the first player makes a move 'X' in any of the remaining eight spaces, then the computer is fixed to move '0' in the center space. In the next decision tree diagram, we will explain how the logic of the algorithm reacts and make decisions for the successive moves and therefore attributing to the No-Loss Strategy.

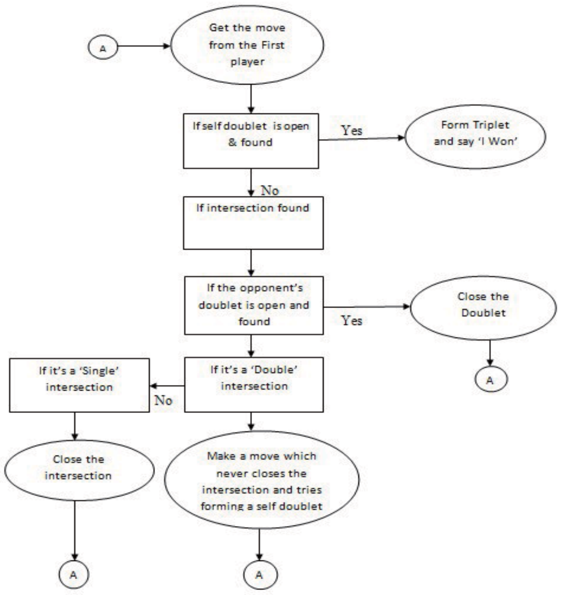

A
Get the move from the First player
If self doublet is open & found
Yes
Form Triplet and say 'I Won'
No
If intersection found
If the opponent's doublet is open and found
Yes
Close the Doublet
A
If it's a 'Single' intersection
No
If it's a 'Double' intersection
Close the intersection
Make a move which never closes the intersection and tries forming a self doublet
A
A

Figure 6: Decision tree representing the successive moves

Before explaining the above diagram, we would like to highlight some of the terminologies we've used throughout:

'Doublet': Doublet is a term which denotes two 'X's or '0's which are placed consecutively or alternatively in a horizontal, vertical or diagonal pattern leaving the third position vacant.

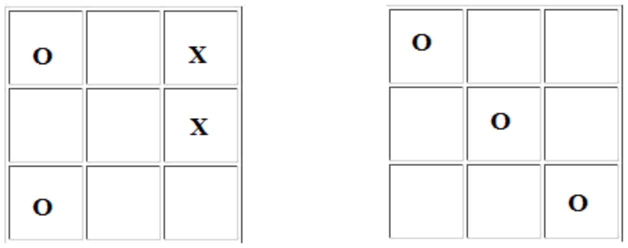


Figure 7: Doublet of X and O & a Triplet.

'**Triplet**': Triplet is a term which denotes three 'X's or '0's which are placed consecutively or alternatively in a horizontal, vertical, or diagonal pattern leaving no position vacant.

'**Single Intersection' :** The intersection of (imaginary) lines formed by two of the moves made by the opponent('X')where one of the intersection line is already blocked by the player(Computer, '0'), the intersection points can be obtained by interchanging rows and columns if the board is assumed to be a 3x3 matrix.

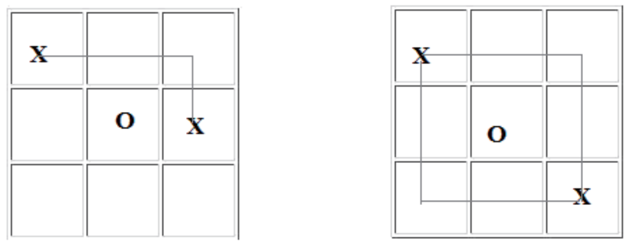


Figure 8: Single & Double Intersection

'Double Intersection': The intersection of (imaginary) lines formed by two of the moves made by the opponent('X'), the intersection points can be obtained by interchanging rows and columns if the board is assumed to be a 3x3 matrix. In proceeding to the first move made by computer, it follows the above mentioned decision tree for further successive moves. Therefore, the computer again gets the move from the opponent and checks through different decision nodes which are as follows:

(1) If a self doublet is found open, it closes the move to form a triplet and say 'I Won'.

(2) If opponent's move forms an intersection, it checks for an open doublet and closes it, or else

(3) If it finds a double intersection then, it makes a move which will never close the intersection and tries forming a self doublet, or else if it finds a single intersection, then it makes a move to close the intersection.

IV. EXPERIMENTATION

In this section, we would like to elaborate on experimentation with the two algorithms in the game of Tic-Tac-Toe and also bring out the comparison between them by Constructing the search trees manually for Minimax and Decision tree selecting random states and then implementing both Algorithms separately in Tic Tac Toe and infer the optimality and the completeness of the solution provided. Compare the Time and Space Complexities.

A. Experimentation with Minimax

We were able to find out that the Minimax algorithm failed to make an optimal move when the opponent makes a sub optimal move almost in 40 different cases. The logic of the game works perfectly only when both players play the game optimally. It's very interesting that in spite of making a sub optimal move, the algorithm is able to provide a 'No Loss Strategy'. But, the total number of moves in which it achieves the terminal state does not seem to be optimal which can be proved through some of the manual experimentation such as constructing the search trees manually for those selected 'special' cases. The diagram shown below clearly explains about the methodology for constructing the heuristics. We used the same heuristics to calculate and find out the trivial states in which the Minimax failed to make an optimal move against a sub optimal player. Some of the cases we analyzed where the Minimax algorithm didn't perform quiet well against a sub optimal player are:

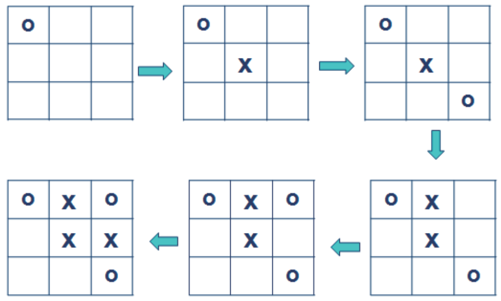


Figure 9: Test case 1

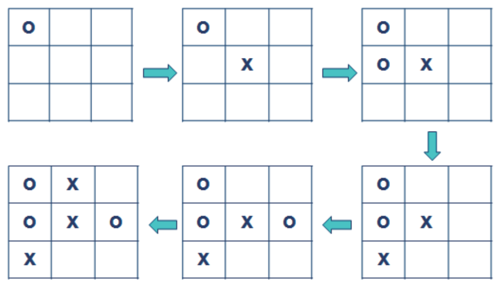


Figure 10: Test case 2

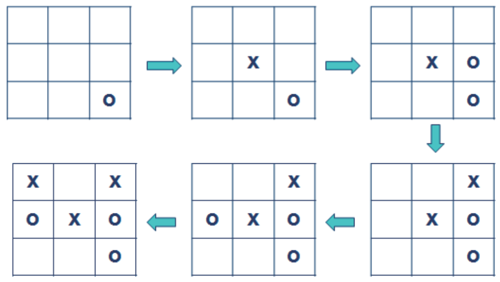


Figure 11: Test case 3

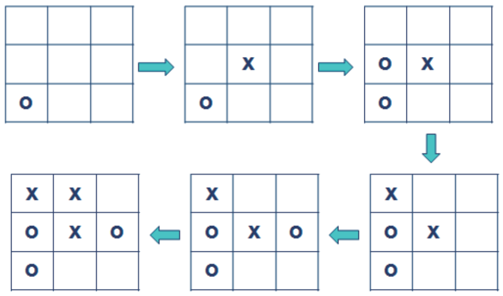


Figure 12: Test case 4

We actually implemented the game of Tic Tac Toe using Minimax algorithm and then used all these test cases for experimentation purposes. Finally, we were able to find out that all these trivial states not only failed when we constructed the search tree manually but, they failed in the real game as well. Henceforth, we were able to verify the logic and functionality of the algorithm. The test cases were used as the bench mark for the next experimentation with Decision trees.

B. Experimentation with Decision Tree

The test cases analyzed in the above section were used as the criterion for developing the game of Tic Tac Toe using Decision tree. The Decision tree was constructed in such a way that it never fails in any of the above mentioned test cases keeping mind that it plays the game optimally in spite of having an optimal or sub optimal player at the other end. We implemented Tic Tac Toe using Decision tree and experimented it with all the 40 test cases. We were successful in proving that it never failed in any of the above cases. The diagram shown below clearly explains one of the sample test cases in Decision tree.

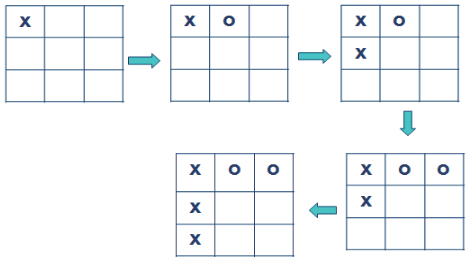


Figure 13: Test case analyzed using Decision tree

It's evident from the above diagram that decision tree makes an optimal move and wins the game in spite of the opponent making sub optimal moves. The Decision tree also performed well against an optimal player. Our main objective is to bring about the failure of the Minimax and prove it the other way with the Decision trees and not to optimize the Minimax algorithm. We also found out that there are some optimization strategies for Minimax algorithm which could be adopted in the near future.

C. Comparison Study

We implemented Tic Tac Toe using Minimax and Decision tree algorithms in 'Turbo C'. We did experiment on both algorithms with all these test cases using which we were able to conclude that Decision tree outperformed the Minimax in the cases where the opponent was a sub optimal player.

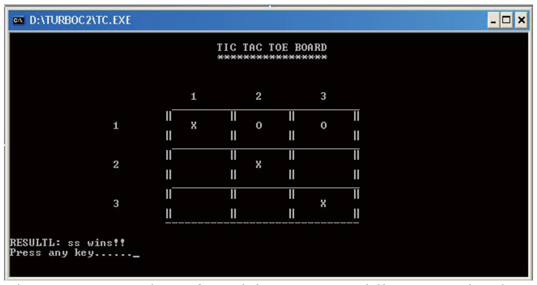

```
D:\TURBOC2\TC.EXE

                              TIC TAC TOE BOARD
                              *****************

                         1         2         3
                    ||        ||        ||        ||
           1        ||   X    ||   O    ||   O    ||
                    ||        ||        ||        ||
           2        ||        ||   X    ||        ||
                    ||        ||        ||        ||
           3        ||        ||        ||   X    ||
                    ||        ||        ||        ||
                    -------------------------------

RESULTL: ss wins!!
Press any key......_
```

Figure 14: Snapshot of Decision tree providing an optimal No-Loss state (Wins in the above example)

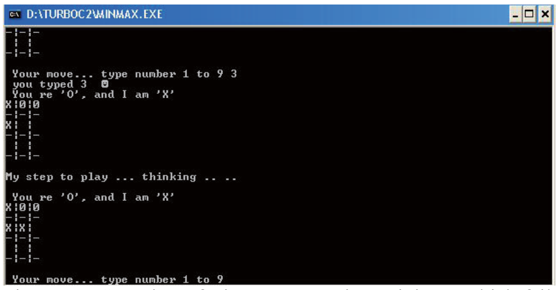
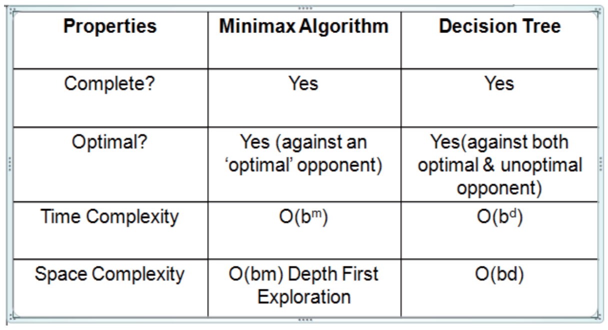

```
D:\TURBOC2\MINMAX.EXE
-¦-¦-
 ¦ ¦
-¦-¦-

 Your move... type number 1 to 9 3
 you typed 3  ☻
 You re 'O', and I am 'X'
X¦0¦0
-¦-¦-
X¦ ¦
-¦-¦-
 ¦ ¦
-¦-¦-

My step to play ... thinking .. ..

 You re 'O', and I am 'X'
X¦0¦0
-¦-¦-
X¦X¦
-¦-¦-
 ¦ ¦
-¦-¦-

 Your move... type number 1 to 9
```

Figure 15: Snapshot of Tic Tac Toe using Minimax which fails to play optimally against a sub optimal opponent

We would also like to bring about the comparisons between Minimax and Decision tree which are as follows:



Properties	Minimax Algorithm	Decision Tree
Complete?	Yes	Yes
Optimal?	Yes (against an 'optimal' opponent)	Yes(against both optimal & unoptimal opponent)
Time Complexity	O(b^m)	O(b^d)
Space Complexity	O(bm) Depth First Exploration	O(bd)

TABLE I:
Comparison between Minimax and Decision tree Algorithm

V. CONCLUSION & FUTURE WORK

Henceforth, it's obvious that Minimax fails to make an optimal winning move in 25-40 cases. The amount of work a Minimax search generates increases exponentially as a move is examined to a greater depth. Also, when exploring the Minimax tree, it sees equally promising nodes which says you can definitely win exploring this move. Presently in such situations, the algorithm breaks the confusion by picking the move with lower position id which might not be an optimal move in order to achieve the No-Loss State. On the other hand, Decision tree algorithm which has a top-down approach will always lead to a terminal state (Win or Draw) by making optimal moves even if the opponent doesn't play the game optimally. In contrast to the Minimax, Decision tree made optimal winning moves when experimented with the same test cases. An optimized approach to reduce the number of branches to be searched by Minimax search algorithm is to implement Alpha-Beta cutoffs in the Minimax search algorithm. It follows the Minimax principle of examining the cost of a player move in depth of the game tree. This approach can however be used only to improve the efficiency of the algorithm. The concept of decision tree which we have introduced here is game specific and it could be applied only to Tic Tac Toe. However, it's much better to develop a generalized approach which could be applied to other games. Therefore, we are planning to extend this research by developing a strategy which could be applied to other games as well.

ACKNOWLEDGMENT

This work was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] Anurag Bhatt, Pratul Varshney, and Kalyanmoy Deb "In Search of No-loss Strategies for the Game of Tic-Tac-Toe using a Customized Genetic Algorithm" ,Pages 889-896, 2008, Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA.

[2] Plamenka Borovska, Milena Lazarova, "Efficiency of parallel minimax algorithm for game tree search", ACM International Conference Proceeding Series; Vol. 285 Proceedings of the 2007 international conference on Computer systems Bulgaria Article No. 14 Year of Publication: 2007 ISBN:978-954-9641-50-9 .

[3] G. Hochmuth. "On the genetic evolution of a perfect Tic-tac-toe strategy", pages 75–82. Stanford University Book Store 2003.

[4] E. Soedarmadji. Decentralized decision making in the game of tic-tac-toe. In Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games, pages 34–38,2005.

[5] An article take from the resource provided by the North Western University, USA www.math.northwestern.edu/~mlerma/courses/cs310-04w/notes/dm-dectrees.pdf

[6] Thomas Hauk, Michael Buro and Jonathan Schaeffer, "*-Minimax Performance in Backgammon", Computer and Games Conference, Ramat-Gan 2004, pg 61-66

[7] Russell Stuart J., Norvig Peter. "Artificial Intelligence. A modern approach.". Prentice Hall, 1995. ISBN 0-13-103805-2.

[8] An Article from the resource provided by the University of California,Berkeley,USA,http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.htm

[9] An article from the lecture notes provided by the University of California, Irvine, USA, www.ics.uci.edu/~dechter/courses/ics-271/fall-08/lecture-notes/6.games.ppt