Dan Scarafoni

2/11/13

Tic Tac Toe AI

Introduction

The goal of this project was to create an artificial intelligence that played Tic Tac Toe against a human player, and played well. The program was written in java consisted of two separate games which operated on different parameters. The first game was a standard game of Tic Tac Toe on a 3x3 board. The second game was a different version of Tic Tac Toe, one with a much more complex state space. For the purposes of the experiment, a 4x4x4 three dimensional game of Tic Tac Toe was used for part 2. This was chosen over nine-board Tic Tac Toe because I am interested in multidimensional computational problems, and the size of the cube could be adjusted more readily (to 5x5x5, 6x6x6, etc...), which allowed for greater testing of parameters.

Tic Tac Toe

Tic Tac Toe is a turn based, zero sum game with perfect information. The game begins with a blank board of three by three space. Two players (represented as X and O) take turns placing markers on the board (on any location) until either the board is full or one player achieves three in a row, column, or diagonal of pieces. The 3D Tic Tac Toe used in the second part operates on nearly identical parameters; players still attempt to achieve three (or, in the case of the 4x4x4, 4). All straight lines in any dimensions are legal. Player X always starts in either case.

Tic Tac Toe can be evaluated as a state search problem. Each state of the board (with the given combination of X's and O's in different positions) represents a separate state for the program. Discounting illegal states (for instance, all O's or all X's), there are 9! states 362880. the 4x4x4 board gave 64! total possible states. Figures 1.1 and 1.2 illustrate the differences between a to dimensional and three dimensional Tic-Tac Toe board respectively.

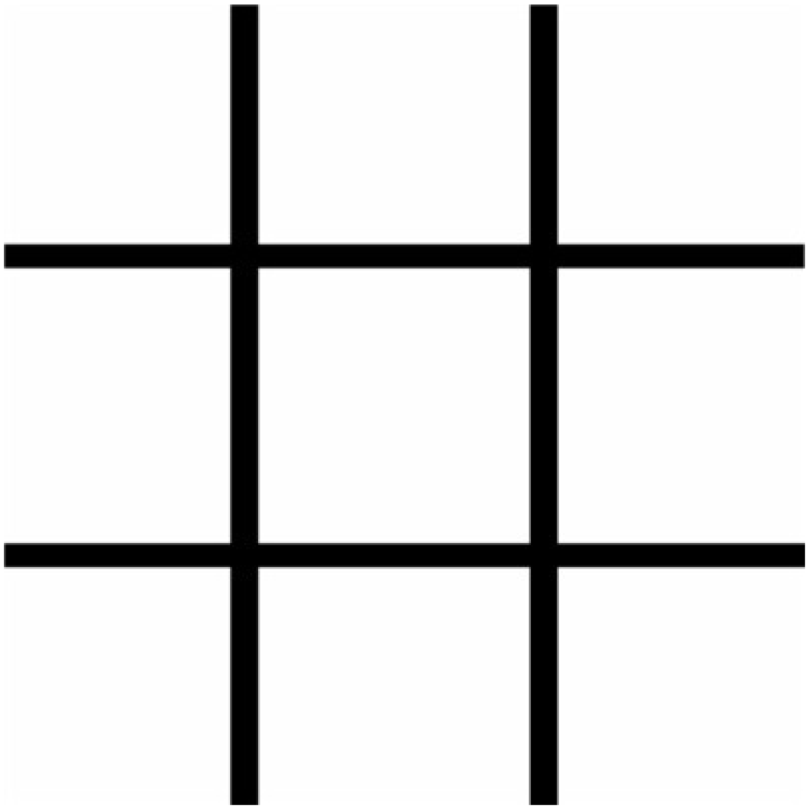

figure 1.1- a standard Tic Tac Toe Board, image from

http://mouserunner.com/MozllaTicTacToe/TicTackBack_400x400.png

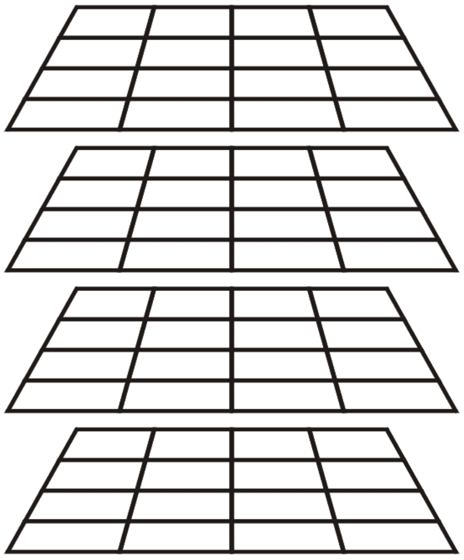

figure 1.2- a 4x4x4 3D board, image from

http://www.printactivities.com/Paper-Games/Tic-Tac-Toe-Games/3D4x4.gif

Formally, the game can be represented as a state based search with the following qualities:

1. States- The arrangement of X's and O's on the board at a given time, as well as whose turn it is.
2. Possible actions- the empty positions on the board where pieces can be placed.
3. Initial State- the empty board at the beginning of the game.
4. Goal state- any state in which there are either three O's or X's in a row on the board

   Transition model- players take turns placing additional pieces on the game map, thus changing the state
5. Transition cost- because the game is adversarial, each player must place their pieces in such a way as to ensure that the goal state will be a victory condition for them, and not their opponent. They must also try to achieve this goal in as few moves as possible.

Algorithms

The standard Tic Tac Toe game could be played using a minimax algorithm. For this algorithm, the computer, at the beginning of its turn, selects all available moves from the current state and proceeds to predict the future actions of the game if these moves are taken. It does so by a series of recursive calls on functions that calculate the best move for the computer (during its future turns), and the best moves for the human (during their future turns).These functions are called Maximize and Minimize respectively.

The recursive calls stop after a terminal is reached on the theoretical board; the computer assigns a utility of +1 to all computer wins, -1 to all human wins, and 0 to all draws. At this point, the program backtracks through the call stack, and the best move for the player or computer (depending on whose turn it is at every level) is chosen. During each's respective turns, the move that benefits the current agent the most is chosen. The best move is chosen by the utility value. The human player is assumed to play for the lowest utility, and the computer player for the highest.

A state tree can be created for all possible moves in the game. In the 3x3 version of the game, the state tree begins with the empty board. From here, there are nine possible child nodes (one for each move). Each one of these has eight children, each of which has seven children, etc. The number of states at each level of the tree m is shown in figure 1.3, and the total number of nodes for a state tree is after a given number of moves m is shown in figure 1.4

$$Nodes(m)=\frac{9!}{(9-m)!}$$

figure 1.3

$$Total\ Nodes(m)=\sum_{i=0}^{m}\frac{9!}{(9-m)!}$$

figure 1.4

Because the total number of moves increases exponentially at each level, the state space become very large and very costly. Both the time and space complexity of the problem is O(n!). The state space was small enough for the 3x3 problem; the minimax algorithm could explore the entire state space in less tan three seconds.

For the three dimensional problem, however, the amount of information to be processed was far to large to be done efficiently. Exact numbers on the efficiency are further discussed in the results section. In order to increase efficiency, alpha beta pruning was used. Because of the extra dimension, the total number of nodes after each move is quite different in the state tree. The number of nodes at a given depth m and the total number of nodes after a given depth m are shown in figures 1.5 and 1.6 respectively.

$$Nodes(m)=\frac{64!}{(64-m)!}$$

figure 1.5

$$Total\ Nodes(m)=\sum_{i=0}^{m}\frac{64!}{(64-m)!}$$

figure 1.6

The technique relies on the a nuance of the minimax algorithm. If a player at node n has a better choice (in terms of utility) at a parent node, then node n will never be played, and thus can be ignored. This formula has can reduced complexity to the square root of its previous complexity.

This technique, however, proved insufficient in order to obtain a reasonable computation time. As such, a limitation to the depth of the search was added. The algorithm was changed from a depth first search to a depth limited search. After a certain depth (three in the soon to be shown test cases), the tree was evaluated as was. Terminal nodes were evaluated as normal, and incomplete game states were evaluated with a utility of 0. From these available states, the best move was chosen. Further the AI's computational time was shortened in most games by the addition of a small heuristic. The program would place a marker in spot 0,0,0 (a corner piece) on its first move if this area was unoccupied. This placement opens the most possible winning conditions for a first move on a blank board, and would drastically cut down on the computational time of the initial placement (which was, by far, the most expensive and time consuming computation).

Results

In both of the models of the game, the AI performed exceptionally well. Figure 1.7 shows the results of twenty games total of the AI versus myself.

wins as O	losses as O	ties as O	average time per move as O (ms)
3	2	5	30.13
wins as X	losses as X	ties as X	average time per move as X (ms)
9	0	1	65.21

figure 1.7

The AI was able to win nine rounds out of ten when it made the first move, and yet was able to tie the game more than anything when it was forced to make the second move. The average time per move doubled between starting second and first. This is logical, as without having to process the first move, the largest state space it will ever have to process is eight levels deep, instead of nine.

The results for the three dimensional cube are even more promising. Figure 1.8 shows the results

wins as O	losses as O	ties as O	average time per move as O
9	1	1	1795.3333333333
wins as X	losses as X	ties as X	average time per move as X
10	0	0	350.2

figure 1.8

The AI clearly outperforms the human in these results, even without knowing the total state tree of the problem. As was seen with the two dimensional game, not having to make the first move radically reduces the average computational time per move. In this case it is the X role that reduces computation time instead of the O role. This can be attributed to the aforementioned heuristic, which removed the most expensive calculation from the program.

One further area of research was trade off between depth of search and move time. In order to answer this question, the two were plotted against one another. The size of the board was also increased (from 4x4x4 up to 6x6x6) in order to show the effect this had on the AI's computational time. This is shown in figures 1.9, 1.10, and 1.11. Note that in the diagram, only the first move of the AI was plotted after the AI was assigned to play X. The small heuristic was removed to reflect the actual computation time of the first move for this experiment. Please note that any move that took more than 2 minutes to finish was judged to be too long as a practical move and represented by a value of -1000 in the charts.

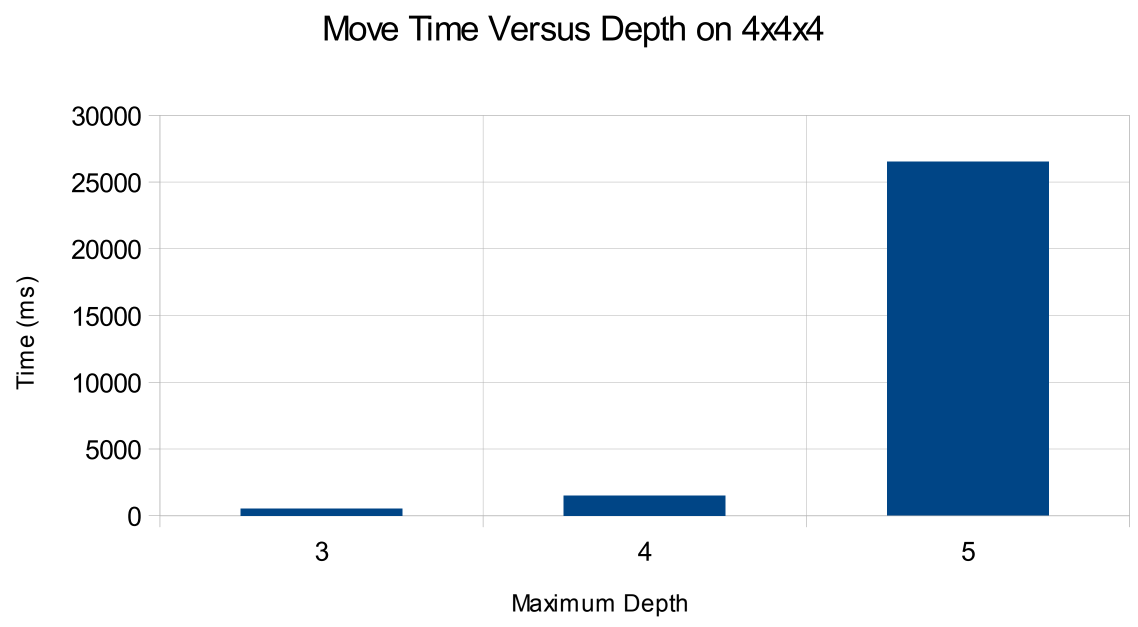

Figure 1.9

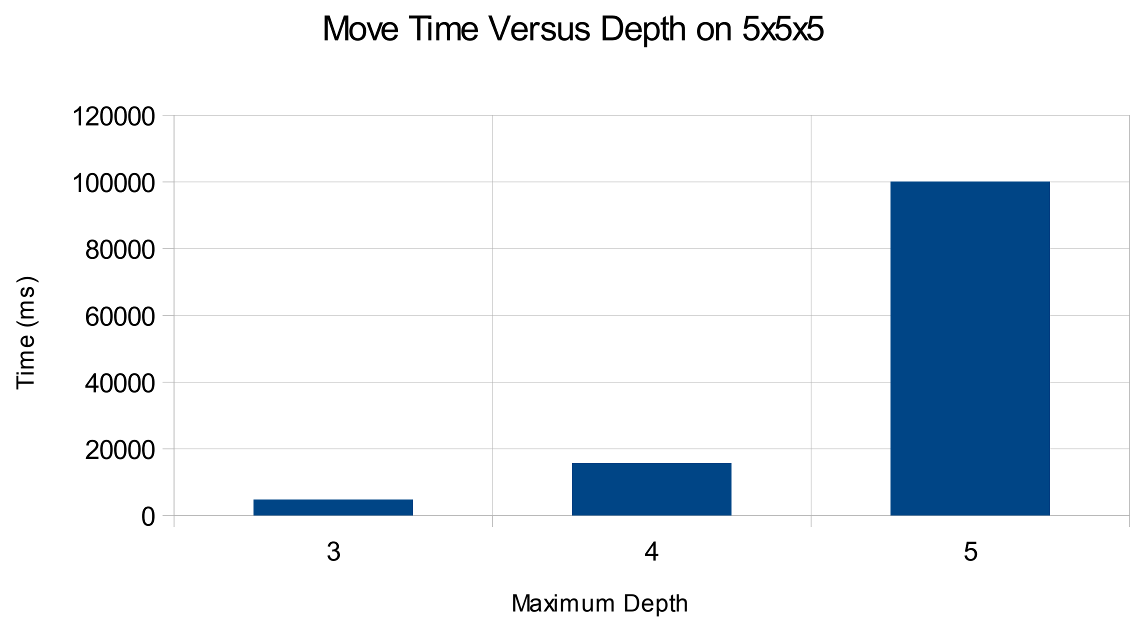

figure 1.10

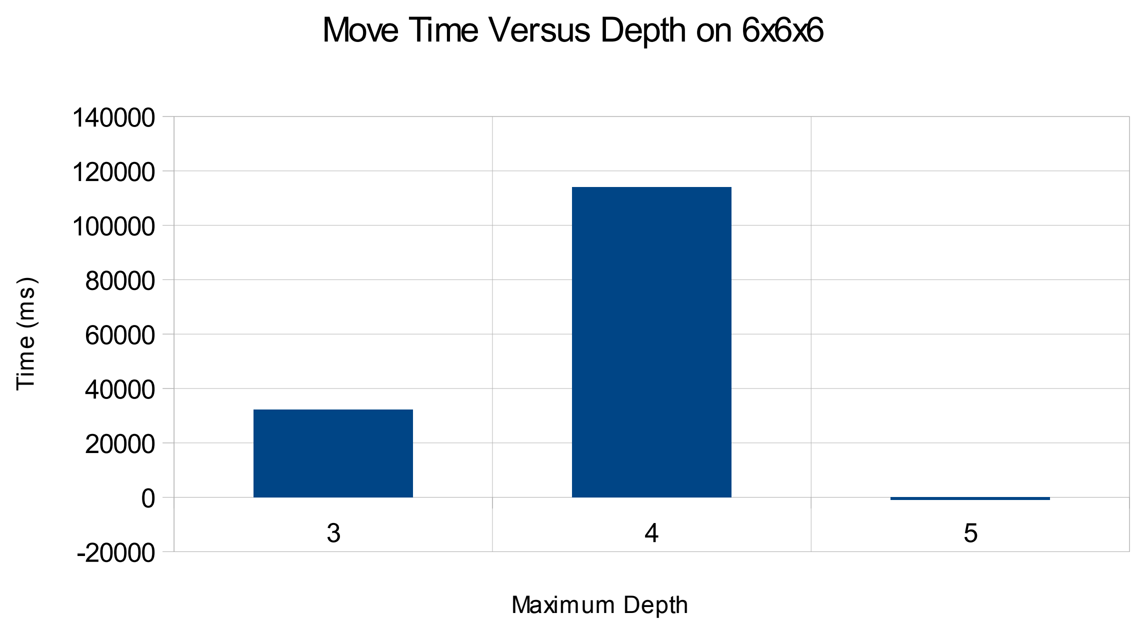

figure 1.11

Conclusions and Discussion

The exponential complexity of the Tic Tac Toe problem created a number of computational problems and limited the evaluation power of the AI's used. Although the AI was able to tie or beat human intelligence in most tests, it was quite clear that it would not suffice for problems of larger complexity without sacrificing the amount of future moves it could predict. This, accordingly, means that the AI would have to sacrifice its ability to plan ahead in order to deal with more complex problems.

Further, there is a need for more rigorous testing of the program versus human AI. The tests in this experiment were few, and only against one individual. In order to more holistically evaluate the effectiveness of the algorithms, this agent should be tested against other humans and other AI.

Finally, there are a number of improvements to the program that could be made. Depth limited searching provides a very black and white and incomplete measure of the value of various moves. The algorithm might be improved by adding in algorithms to compute the utility of moves that do not terminate the board, but instead allow the player to place two or three pieces in a row, or prevent the opponent from doing the same. Further, more research could be done into the placing of the initial piece on the blank board. Although the corner piece does open the most winning situations (on a 4x4x4 board), it may not be as good of a choice in the long run, as it may open up less winning possibilities later in the game.