Grace Handler and Konstantine Mushegian
Professor Majercik
Artificial Intelligence
October 5, 2016

Minimax for Connect-4: Technical Report

*NOTE: We based our code off of Chapter 5 in the cited textbook and adapted code from that provided by Professor Majercik.

Description of Connect-4

Connect-4 (known as Gravitrips in Soviet Union) is a two-player connection game in which players take turns dropping colored discs from the top into a vertically suspended grid. In the provided implementation the grid is 8 columns wide and 6 columns tall. The pieces are guaranteed to fall straight down as far as possible until they either hit the bottom or another piece. The objective of the game is to connect one's own discs of the same color next to each other vertically, horizontally or diagonally before the opponent. Once one of the players accomplishes to place four of their pieces next to each other, the game is over.[1]

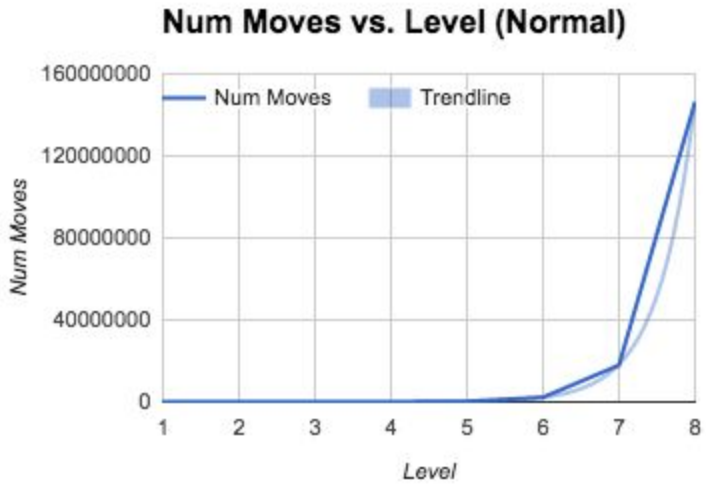
How Minimax Relates

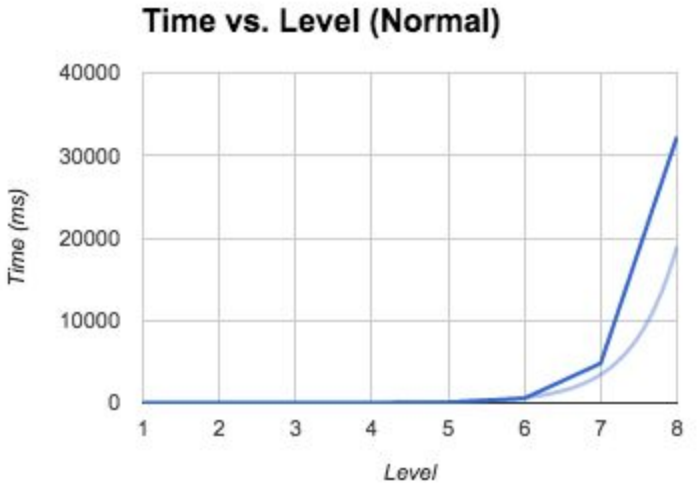
A game of Connect-4 can be boiled down to a back and forth between players. This is where Minimax comes in handy, since the general idea of Minimax is also a back and forth between players, where each player tries to pick a move that will that will maximize the chance of winning. The general Minimax algorithm will explore all possible moves at each step until it reaches the terminal state, i.e. until it reaches the end of the game. The highest value will then be passed up the chain and the move with the highest projected value will be executed. However, computing all the possible moves is expensive. Alpha-Beta pruning is a way to speed up a classical Minimax algorithm by reducing the total number of explored moves. A Minimax algorithm that makes use of alpha-beta pruning explores fewer possible states by keeping track of the worst and best case scenarios and ignoring everything that is worse than the worst case and also worse than the best case. Minimax with alpha-beta pruning returns the same moves as Minimax without alpha-beta pruning, however it does so significantly faster due to the reduced amount of total states explored.[2]

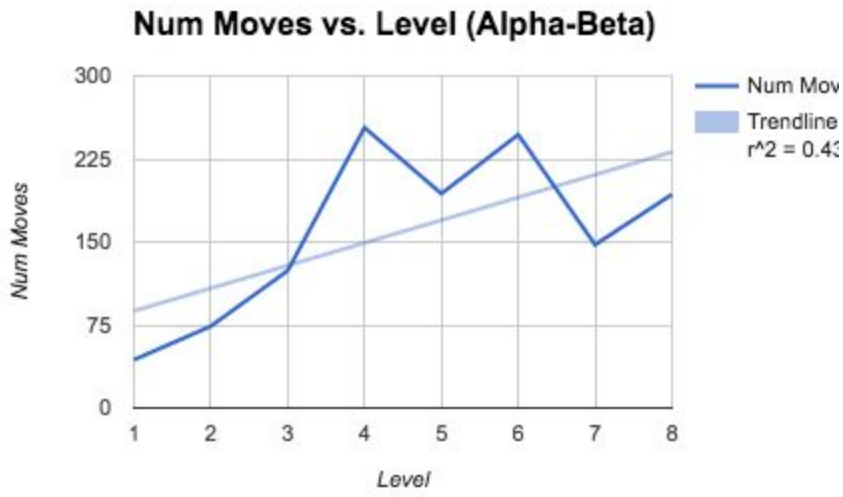
[1] https://en.wikipedia.org/wiki/Connect_Four
[2] *Artificial Intelligence: A Modern Approach*, Stuart Russel and Peter Norvig

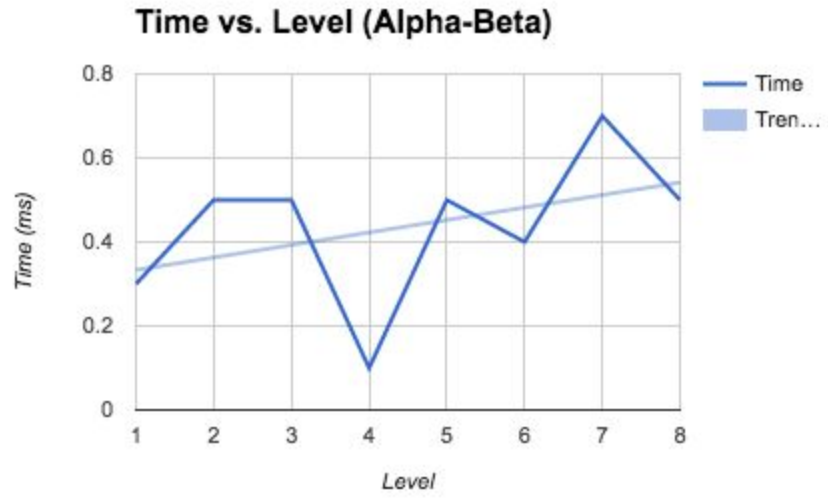
Our Data

Game (Normal)	*Level 1*		*Level 2*		*Level 3*		*Level 4*		*Level 5*		*Level 6*		*Level 7*		*Level 8*	
	Num Moves	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*
Average	*68.6*	*0.3*	*558.5*	*0.6*	*4618*	*1.9*	*36748.7*	*10.5*	*298070.4*	*79.1*	*2304582.4*	*550.9*	*17873944.6*	*4834*	*145109214.3*	*30781.6*
Game (Alpha-Beta)	*Level 1*		*Level 2*		*Level 3*		*Level 4*		*Level 5*		*Level 6*		*Level 7*		*Level 8*	
	Num Moves	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*	*Num Moves*	*Time (ms)*
Average	*44.2*	*0.3*	*72.1*	*0.5*	*122.6*	*0.4*	*232.6*	*0.1*	*204.7*	*0.3*	*245.9*	*0.5*	*146.5*	*0.6*	*193*	*0.5*









Discussion of results

For both normal minimax and alpha-beta pruning, both time and the number of moves increase as the levels increase. In the case of normal minimax, the increase is exponential; whereas, in the case of alpha-beta pruning minimax, the increase is essentially linear. It is important to take into account that the program reported only whole numbers, so for the alpha-beta pruning all the reported times were either 0 or 1, but likely varied more than the data shows. This is also true of the first few levels of normal minimax, particularly in comparison to what the time becomes for level 8. Thus, these data visualizations should only be interpreted for the general, overall trends of number of moves and time per type of algorithm and the type of trends per type of algorithm. Overall, alpha-beta pruning was markedly faster than normal minimax, because it required trying markedly fewer moves each time it expanded. This makes sense since alpha-beta pruning is an optimization of normal minimax.

Answers to Questions

a) The heuristic function computes the number of two, three and four in a row discs that each player has. Let's assume we are computing the strength for Player 1. If Player 1 has four in a row, then its strength is the maximum value that an integer can hold in Java, if the second player has four in a row, then its strength is zero. If neither player has four discs in a row, then the player's strength is computed by the equation below:
$strength = NumberOfTwoInARow^4 + NumberOfThreeInARow^{32} + NumberofOneInARow$

[Note: i << j means i * 2^j]

b) We do not think that left to right or right to left would make any difference. We do however think that an ordering based on the number of discs in each column might be better here, since we would first explore the columns that have a statistically higher chance of taking us to the terminal state after several (or one) moves.

The ordering in Minimax without alpha-beta pruning would not matter as it would explore every possible move until it hits a terminal state at the end of every possible move.

In case of Minimax with alpha-beta pruning ordering could matter as it does not explore each possible move down to the terminal state; it only explores each possible move down to a certain depth. In Minimax with alpha-beta pruning it would make more sense to start searching in a column that has the maximum number of discs in it, or in a column that has the most number of discs in columns adjacent to it. If it were to explore the moves in this order we would be able to find the children with the highest values earlier on, because starting in a clustered place would be more likely to lead us to the winning/losing situation in a small number of back-and-forth moves.

c) The value of an unnamed constant on line 87 of `C4Board.java` should be 69. It would be nice to have a named constant since we would not have to look for a stray unnamed constant on line 87 in some file of the codebase. We obtained the number 69 by counting all the possible groups of 4 slots.