COMP 4106 Project Final Report

By: Chris Dufour

#100854402

Connect Four – Multiplayer AI Technique Comparison

Problem Domain

Connect Four is a classic game that is played on a 6 high, 7 wide grid and 2 players take turns placing colored pieces into the top of a column. After being placed the piece falls to the lowest available spot; if there is a piece below them it sits on top of it.

The game is won when a player manages to line up 4 of their own pieces in a consecutive pattern, either horizontally, vertically or diagonally. Players can prevent each other from winning the game by playing pieces so that they interfere with their opponent's attempts to line up their pieces. In this way it is similar to more complicated games such as chess: you must balance making moves that bring you closer to the goal with moves that prevent the opponent from winning the game. It is also entirely possible to use your opponent's attempts to line up their pieces to your advantage, working their past and present moves into how you will ultimately beat them.

According to the On-Line Encyclopedia of Integer Sequences there are, in total, 4,531,985,219,092 different states that the game can take.

Motivation:

The motivation behind this project is that I wanted to build an AI that can play a game against a human opponent. Many classic games are based on luck, and as such do not make good candidates for an AI to play. Connect Four is a game that can require a great deal of thinking and strategy to overcome an opponent, while being entirely devoid of luck. While there is a method to play perfectly very few, if any, human players would be able to apply it as it requires memorization of a large number of state spaces (with the longest game being won on the 41st move).

AI Techniques and Algorithms:

Minimax Search:

The basis for the searching techniques I used was the Minimax Search. It represents the choices the program can make as leaves on a tree graph, representing the current state of the game as the root of the tree. From those decisions it evaluates the states that can result from those states and determines the best moves to make assuming that the opponent will also make what it believes to be the best possible moves. When encountered with multiple choices with the same heuristic value it will choose randomly between them, preventing it from playing the same moves every time.

Alpha-Beta Pruning:

As an extension to Minimax Search this technique allows for much faster computation of ideal states by decreasing the number of nodes that are evaluated. It does this by not evaluating states that are a result of a state that it evaluated earlier and found to be worse than another state resulting from the same decision. While it is capable of greatly improving calculation speed it can fail to improve the result if the opponent performs a decision it skipped and eventually results in an even better state. This allows for deeper searches to be performed as the amount of time saved makes up for the increased depth.

Quiescence Search:

Quiescence search improves upon Minimax search by further evaluating what it determines to be ‘noisy’ moves, those being moves that have a strong effect on the game. By further evaluating these states beyond the normal depth limit it allows for more informed analysis of potentially game changing situations.

Transposition Tables:

Transposition tables are a technique where the ideal move for a position is calculated ahead of time and stored in a table. This allows for much faster computation of moves as it doesn’t have to evaluate the subtree of a move unless it has never been encountered before.

The algorithm to generate the table is based off of the Minimax search in that any state it doesn't know it will calculate and remember for next time, allowing it to dynamically build a database of which states warrant which moves.

Design Choices:

Program environment:

The program was implemented in Java, partially due to my familiarity with it but also it allows for good logging of memory usage and calculation time for statistical purposes.

Connect Four Game:

The game was implemented as a stateful object. The object contains the game board and information such as who's turn it is and what the current game state is (Player 1 has won, Player 2 has won, Tie or game is running). It also contains some values such as number of turns passed and how many pieces are in each column to speed up calculations.

AIs:

The AIs work on a common interface that allows them to be used abstractly and gives the ability to easily implement new AI types or create modified versions of existing ones. This allows for easy comparison and implementation of the different search techniques.

Heuristics:

The heuristics are implemented as part of the AIs, allowing for multiple Heuristics to be used with the same AI type. The heuristic that I used awards points for how close the current player is to getting a four in a row and removes points for how close the opponent is for getting one.

Results:

When tested against the base Minimax search the results of some of the applied techniques differed. These are the results of 100 trials.

Minimax

Win/Loss/Tie Ratio	Wins	Losses	Ties
As First Player	39.0%	42.0%	19.0%
As Second Player	42.0%	39.0%	19.0%

Total Node Count	Average	Min	Max
As First Player	23079	18323	32894
As Second Player	21729	17163	32469

Total Calculation Time	Average	Min	Max
As First Player	70ms	51ms	313ms
As Second Player	63ms	46ms	108ms

With Alpha-Beta Pruning

Win/Loss/Tie Ratio	Wins	Losses	Ties
As First Player	52.0%	37.0%	11.0%
As Second Player	61.0%	33.0%	6.0%

Total Node Count	Average	Min	Max
As First Player	78019	32892	146422
As Second Player	84638	42008	142280

Total Calculation Time	Average	Min	Max
As First Player	216ms	88ms	430ms
As Second Player	236ms	109ms	444ms

Quiescence Search

Win/Loss/Tie Ratio	Wins	Losses	Ties
As First Player	67.0%	22.0%	11.0%
As Second Player	54.0%	34.0%	12.0%

Total Node Count	Average	Min	Max
As First Player	614020	343709	1053469
As Second Player	624770	385739	900020

Total Calculation Time	Average	Min	Max
As First Player	2070ms	1129ms	3796ms
As Second Player	2087ms	1165ms	4116ms

Transposition Tables

Win/Loss/Tie Ratio	Wins	Losses	Ties
As First Player	43.0%	56.0%	1.0%
As Second Player	33.0%	67.0%	0.0%

Total Node Count	Average	Min	Max
As First Player	20660	2587	30218
As Second Player	21533	16857	29918

Total Calculation Time	Average	Min	Max
As First Player	59ms	8ms	92ms
As Second Player	63ms	48ms	122ms

Minimax:

With the Minimax search as our baseline we can see that the second player has a slight advantage against the player going first. The number of nodes is fairly consistent, but this

comes as no surprise as unless a column is full there will generally be 7 possible states for every node. The calculation time is also consistent and will prove to be a good baseline for how long a calculation will take.

Alpha-Beta Pruning:

The Alpha-Beta pruning search had a higher max depth cap then the other searches in order to take advantage of its reduced calculation time and speed. While the node count is roughly 3 times that of Minimax and the calculation time about double it allows it to achieve a better win ratio against the basic Minimax Search.

Quiescence Search:

While this search was more successful at beating the basic AI it suffered much longer calculation times and massive node counts. It is possible to reduce the depth of the extra length on noisy states, but this will take away the advantage of the search. It is likely that the AI is too sensitive to what is noisy and what isn't (in this case it considers a state noisy when a player can create a four in a row on that state), however it is difficult to determine what should be considered a noisy state as four in a rows can be found at relatively shallow depths.

Transposition Table:

The use of a transposition table has sped up the calculations to a notable degree and reduced the number of nodes required to come to the same result as Minimax search. The win ratio has been slightly affected, but this may be negligible. It is important to note that the transposition table both relies on the heuristic I developed, and may be subject to errors as well as that the table it's self has roughly 5000 states memorized, requiring that any new ones be calculated.

Possible Enhancements:

While the results I gathered are generally useful I was limited in the fact that the searches had to be fairly shallow to run quick enough for a human to play against them (in that there is no significant waiting between moves). In order to improve the results and obtain a larger sample set I would need to run the program on a computationally faster device than my personal computer, the ideal in this case being a supercomputer.

Because of the computation limitations I wasn't able to attempt to determine the perfect play solution, as it requires roughly 41 turns to be analyzed, which is way beyond my personal capabilities. Calculating for every state space is also out of the question as even only using 8 bits to represent a state would require about 5 terabytes of data to store all the states.

Other enhancements I could apply would be to improve the heuristic values of how valuable a state is depending on the positions of pieces. Improving what Quiescence Search considers noisy and quiet would also improve its calculation time and reduce the number of nodes that it has to evaluate.

References:

Adams, Ron, Erik Ibsen, and Chen Zhang. "A Connect Four Playing AI Agent: Algorithm and Creation Process." (2003).
http://a-team-connect4.googlecode.com/svn-history/r4/trunk/Algorithms/Connect4/SmartConnectFour-Final_Paper-v1.0.doc

Lam, Jenny. "Heuristics in the game of Connect-K."
http://www.ics.uci.edu/~jlam2/connectk.pdf

Sarhan, Ahmad M., Adnan Shaout, and Michele Shock. "Real-Time Connect 4 Game Using Artificial Intelligence." *Journal of Computer Science* 5.4 (2009): 283.
http://thescipub.com/abstract/10.3844/jcssp.2009.283.289

Appendix

The program was coded in Eclipse and on Windows 7, and has not been thoroughly tested on other platforms. The program is command line based and has no graphical user interface beyond that.

How to run the program:

1. Switch Eclipse's workspace in to the folder where the project files are located. If it does not appear go to File->Import->Existing projects into workspace and import it.
2. Press Run
3. Follow the instructions in the command prompt.

If you have any issues running the program please contact me:

christopherdufour@cmail.carleton.ca