* A.I. Connect-Four *

Abe Karplus
Science Fair 2010

Abstract

I am using the game Connect-Four to study artificial intelligence, a field in computer science. I wanted to determine the effect of ply depth (the number of plies, which are turns by a single player, that the program looks ahead) and the effect of getting the first move on the chance of winning. To do this, I wrote some computer programs in C to simulate games between computer-controlled players.

I found that increasing the ply depth increases the chance of winning, as does getting the first move. I found that the effect of a single ply-depth increase was greater than the effect of getting the first move. I found that when two identical players faced off, there would be more draws if both had even ply depths.

Table of Contents

Introduction

Artificial intelligence is an important and growing field of study, with practical applications ranging from search and rescue to speech recognition. One good way to study A.I. is through games. These provide a simplified world, so that most of the effort of scientists can be devoted to developing the A.I. and not to constructing robots or simulating the world.

I chose Connect-Four as the game I will study for many reasons. The rules of it are simple, unlike, for example, chess, and so I do not have to devote much effort to the simulation. There are a relatively small number of possible moves at any given point in the game, limiting combinatorial explosion. However, the game still allows a rich depth of strategy, unlike games such as tic-tac-toe.

Artificial Intelligence

Artificial Intelligence, or A.I., is the study of creating intelligent-seeming behavior in computers and robots. Computers excel at fast computation and systematic, repetitive tasks. A.I. researchers make use of this with programs that go through many possible options and select the one that seems best. There are several types of A.I., such as search (which is what this project is concerned with), machine learning, and pattern recognition.

Search attempts to find whichever option fits a certain criterion best. The way I have been implementing search is through the min-max function, which simulates alternating plies where one side selects the maximum value option and the other side selects the minimum. In game algorithms, a *ply* (plural *plies*) is one turn by one player, as opposed to a *round*, which is one turn for each player. A simple min-max algorithm stops looking ahead at a predetermined number of plies, known as the *ply depth*.

Connect-Four

Connect-Four is a game for two players, where the object of the game is to get four or more pieces of your color in a line vertically, horizontally, or diagonally. The game board is a grid of spaces for pieces, six high and seven wide, as in Figure 1.

Figure 1: The Game board

Pieces are inserted at the top of the game board and fall to the lowest open level in their column. Players attempt to get four pieces in a row while blocking their opponent from doing the same. Whoever gets four of their pieces in a line first wins, but if the board fills, it is a draw.

Perfect Players Exist

One approach to creating a game-playing program is to precompute the best possible move in any given situation and store this information in a hash table. The player then merely looks up in the table what move to make. This creates very fast players that, if a complete table has been precomputed, never make mistakes. This approach only works for fairly simple games, and it is not useful in the real world. If creating an A.I. program for the real world, one does not have the luxury of precomputing all possibilities. Instead, the program has to identify the options, predict what will happen for each potential choice, and identify the best outcome.

The game of Connect-Four has been solved completely, with results that show an advantage for the first player. If the first player moves in the center, they can force a win. If they begin on either adjacent square, the second player can force a draw, and if they begin on any other square the second player can force a win.

Even though perfect players exist for Connect-Four, the game is still useful for studying search algorithms.

Hypotheses

I have several predictions about the outcomes of my experiments. I predict that increasing the ply depth of a recursive search for a given evaluation function will increase the chance of winning. I further predict that going first in a game will give a slight advantage to a player—however, I do not believe that this will be enough to offset a difference in ply depth between players. I hypothesize that fevala will do better than wineval, but not by a large amount.

Program Description

I wrote all the code for this project in C. I chose C for two reasons: I already knew the language, having used it for my science fair project last year, and it is very efficient, so that my experiments would not take too long. All the programs for this project are in the Programs folder.

The program is divided into several files. The `connectfour` file deals with running the game or games, including interfacing with the player functions and processing command-line arguments. The `board-disp` file deals with displaying the board for human players. The `boardcontrols` file deals with the data representation of the board. The `recurse-player` and `randperm` files contain the skeleton of a player function (`recurse_play`). All the players except `p-human` call the `recurse_play` function.

recurse-player

The `recurse_play` function is a min-max algorithm. Given a board, a ply depth, and an evaluation function, it returns a structure containing which move to make and how good it considers that move. It tries playing each of the seven possible moves in a random order provided by the `randperm` function. For a one-ply search, it calls an evaluation function on each move and chooses the highest value as the move to return. If the ply depth is higher, it swaps `'X'` and `'O'`, calls itself recursively (reducing the ply depth by 1) to simulate the opponent's play, and uses the return values to decide its move.

One early problem with the algorithm was that it did not distinguish between immediate and distant wins or losses. With wins, this is not a problem, since it will always take forced wins, if any are available. It did not attempt to delay losses, even though doing so could give its opponent more chance to make a mistake. I solved this problem by adding a “decay” to the algorithm, so that it returns 0.95 times the value of the best move in its return structure, thus favoring quick wins and delayed losses.

Figure 2 is a simplified diagram showing the process of the `recurse_play` algorithm. The boxes represent calls to the algorithm and the numbers outside of boxes, the results of calls to the evaluation function (here `wineval`). In a box, the first field (`'X'` or `'O'`) displays whose turn is being simulated, the next is the value of the move returned, and the last field is what moves the function might return. The diagram is simplified to only three possible moves (A, B, C) instead of the seven of the full game, and it only displays a 3-ply player.

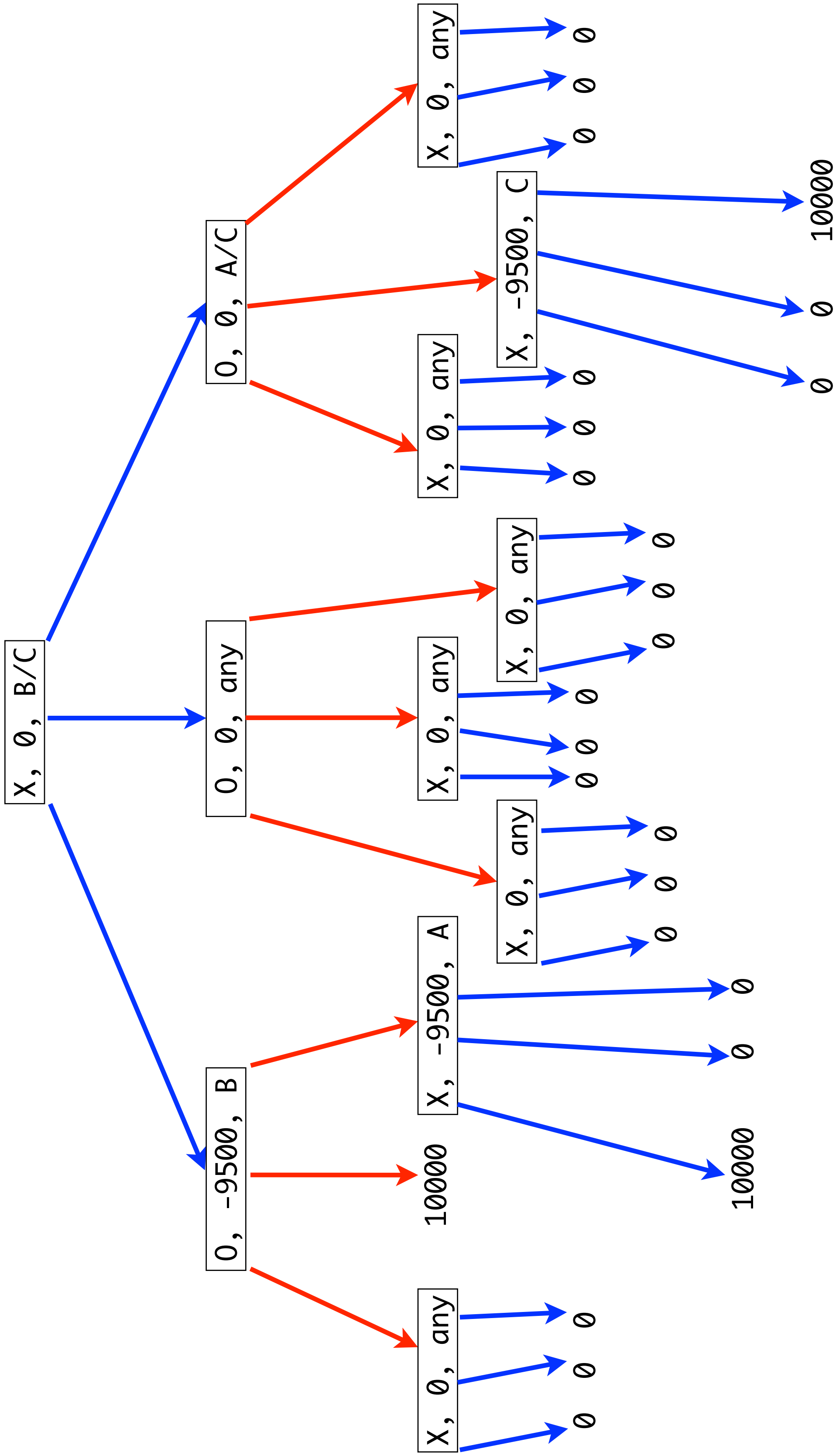

Figure 2: The `recurse_play` *algorithm*

boardcontrols

The `boardcontrols` file contains many functions, all of which manipulate or inspect the representation of the board. The game board is currently represented as an array of columns, each of which is an array of characters. The characters used are `'X'`, `'O'`, or `' '` (space). Here is a list of the `boardcontrols` functions:

>> `clearboard` removes all pieces from the board.

>> `printboard` displays the board on the screen. It uses ASCII graphics—the setup shown in Figure 1 would look like Figure 3. If either player is human, this function is not used (see `board-disp` below).

>> `playtoboard` places a piece for the X player into the lowest empty space on the given column.

>> `unplayboard` removes a piece for the X player from the given column.

>> `invertboard`, one of the most frequently used, swaps the X pieces and the O pieces. Almost all the other `boardcontrols` functions are simplified by assuming that it is the X player's turn.

>> `is_boardfull` determines if the game is a draw.

>> `is_movewin` determines if the X player has just won by playing in the given column.

```
0 1 2 3 4 5 6
^ ^ ^ ^ ^ ^ ^





      O
      X O
^ ^ ^ ^ ^ ^ ^
```

Figure 3: ASCII graphics

Player Functions

Each different player function has its own file: `p-human`, `p-random`, `p-feval-1`, and `p-wineval`.

The human player takes a number from `stdin` (standard input) and returns that as its move. Thus, a person can play by typing numbers.

The random player calls `recurse_play` with evaluation function `randeval` (which returns a constant), so the recursive algorithm is used simply as a random number generator.

The `wineval` player calls `recurse_play` as well. The `wineval` player provides the evaluation function `wineval`, which calls `is_movewin` to determine whether the move just made won the game.

Finally, there is the feval class of player functions, which currently includes `fevala`, `fevalb`, and `fevalc`. Theses players work by looking at each adjacent set of four cells (or "four"), assigning it a value based on what pieces it contains, and returning the sum of the values of the "fours". A four can be blocked (containing pieces from both players), empty, a win (four X pieces), O1 to O3 (one to three O pieces), or X1 to X3 (one to three X pieces). The three feval functions differ only in the values they assign to each possibility (with win causing an immediate return of 10000).

Player↓	Blocked	O3	O2	O1	Empty	X1	X2	X3
fevala	0	-3	-2	-1	0	1	2	3
fevalb	0	-40	-15	-3	0	2	13	22
fevalc	0	-600	-60	-12	-2	10	50	500

This table shows the values assigned by each function. I came up with the constants for `fevalb` and my dad chose those for `fevalc`.

connectfour

The main program serves as a wrapper allowing the user to control which players play against each other, for how many games, whether the players alternate first move, and whether to display the board after each move. It also keeps statistics on wins by each player, draws, number of moves by each player, and time taken. I wrote it to be controlled by command-line arguments, so that automating the experiments would be easier. Automation was done with Makefiles and tcsh scripts; see Appendix 2 for the Makefiles and Appendix 3 for a sample tcsh script.

board-disp

At school science fair, I received some complaints about the display of the board. The computer frequently moved too fast for humans playing against it to easily determine where it last moved. I decided to have the display highlight where the last move was made. Unfortunately, using basic ASCII graphics does not permit any "special effects" like highlighting or color. Therefore, I created a new program for displaying the board using `ncurses`. `Ncurses` (short for "new cursor optimization") is a programming library that allows textual graphics with color, bold, italics, some non-ASCII characters, and more effects. Figure 4 is a screenshot from the display program I wrote using `ncurses`.

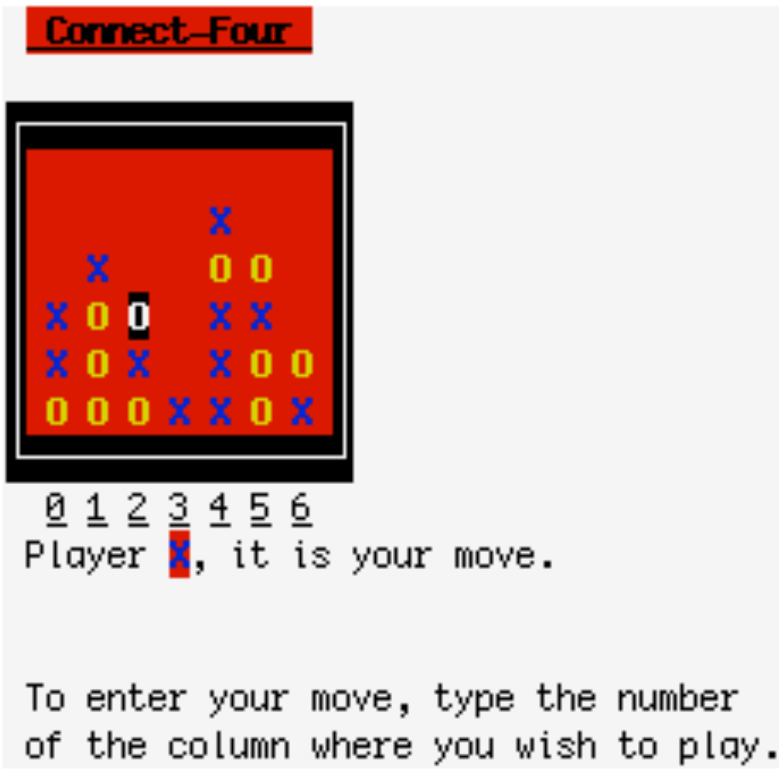

Figure 4: Ncurses display

Due to some problems with screen output by other functions, I wrote the `ncurses` program as a separate file, compiled separately and called by the main program in the `connectfour` file by means of a `system` command. The main program will call the `board-disp` program if no players are specified in the command line. I also have the `board-disp` program set up so that it will ask the player who they want to play against (another person or one of four difficulty levels on the computer).

Experiments

I did seven experiments in this project. For the first experiment, I looked at the difference between `wineval` players with different amounts of lookahead (ply depth). A short tcsh script ran all players (`random` and `1ply_wineval` through `5ply_wineval`) against all players. Each pair of players played 1000 games, and X and O alternated who moved first to eliminate any first-move bias.

The second experiment ran each `wineval` player against every other `wineval` player including itself, but with X always going first. This investigated both the effect of first move on its own (when X and O are the same) and in conjunction with differences in ply depth. My experiments only went as far as 5 plies of lookahead for two reasons. When I ran the first experiment, I had only written player functions for 1 through 5 plies. I later changed the code so that the user can specify up to 9 plies. Also, each additional ply takes 7 times as long (a phenomenon known as *combinatorial explosion*), and the experiment took quite long enough at only 5 plies.

The third experiment was like the first, only for the `fevala` player.

The fourth experiment was like the second, only for the `fevala` player.

The fifth experiment ran each `fevala` player (1 to 5 plies) against each `wineval` player, with first move alternating.

The sixth experiment ran each `fevala` player against each `wineval` without first move alternation.

The seventh experiment was a test of `fevalb` and `fevalc`. It tested them against each other, `fevala`, and `wineval`, though only with identical ply depths and first move alternation for 100 games.

Results

See Appendix 1 for a complete set of all data from these experiments.

Wineval: Alternating First Move

The value shown is wins by the X player plus one-half of the draws from 1000 games, when X and O alternate who moves first.

O↓ X→	random	1ply	2ply	3ply	4ply	5ply
random	498					
1ply	246	488				
2ply	58	110.5	501			
3ply	49.5	101.5	345.5	517		
4ply	12	24	218.5	287	515	
5ply	9	21.5	199	260.5	399.5	481

See Figure 5 for a graph.

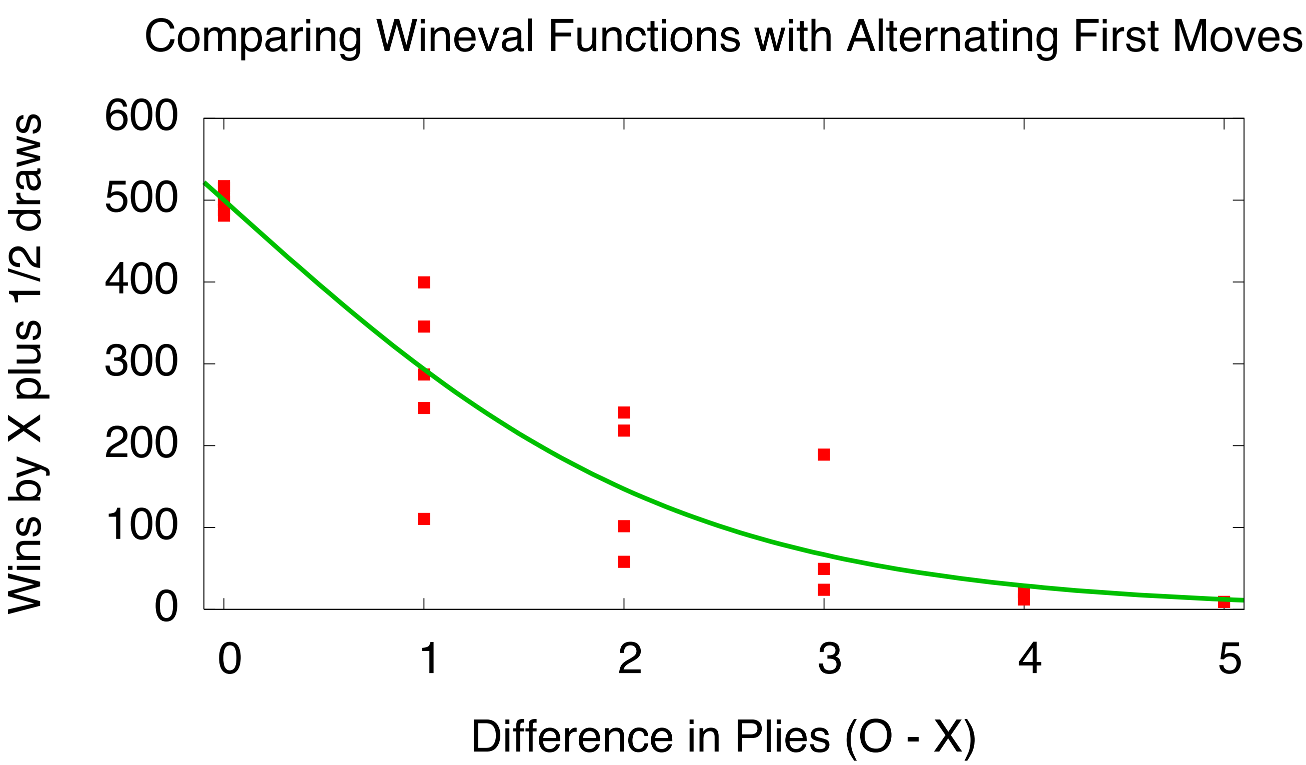

Figure 5: The difference in plies makes a large difference in who usually wins—specifically, an increase in ply depth causes a larger percentage of wins. The curve shown is a logistic function fitted to the data by gnuplot, see sample script in Appendix 3.

One other phenomenon I noticed in the data was the pattern of draws when the two players were identical:

1ply	2ply	3ply	4ply	5ply
0	120	28	158	46

It appears that the number of draws alternates with the ply depth, with even ply depth producing many more draws. This makes sense when we consider what ply depth means. Looking an odd depth ahead means that the player is better at offense than defense. An even depth, with equal ability at offense and defense, means that more attempts will be blocked, causing the board to fill up and increasing the likelihood of a draw.

Wineval: X Moves First

The value shown is wins by the X player plus one-half of the draws from 1000 games, when X always goes first:

O↓ X→	random	1ply	2ply	3ply	4ply	5ply
random	557	832.5	953.5	973.5	991	987
1ply	318.5	595	932.5	925	988.5	987.5
2ply	74	151.5	525.5	679.5	792.5	817.5
3ply	48.5	113.5	361.5	558.5	759	767
4ply	12.5	21.5	233	293.5	537.5	651
5ply	16.5	24.5	220.5	296.5	416.5	528.5

The numbers on the main diagonal are always greater than 500, meaning that going first increases the chance of winning. The cells directly below this show that going first does not offset the disadvantage of being one ply behind, as all those numbers are substantially under 500. See Figure 6 for a graph.

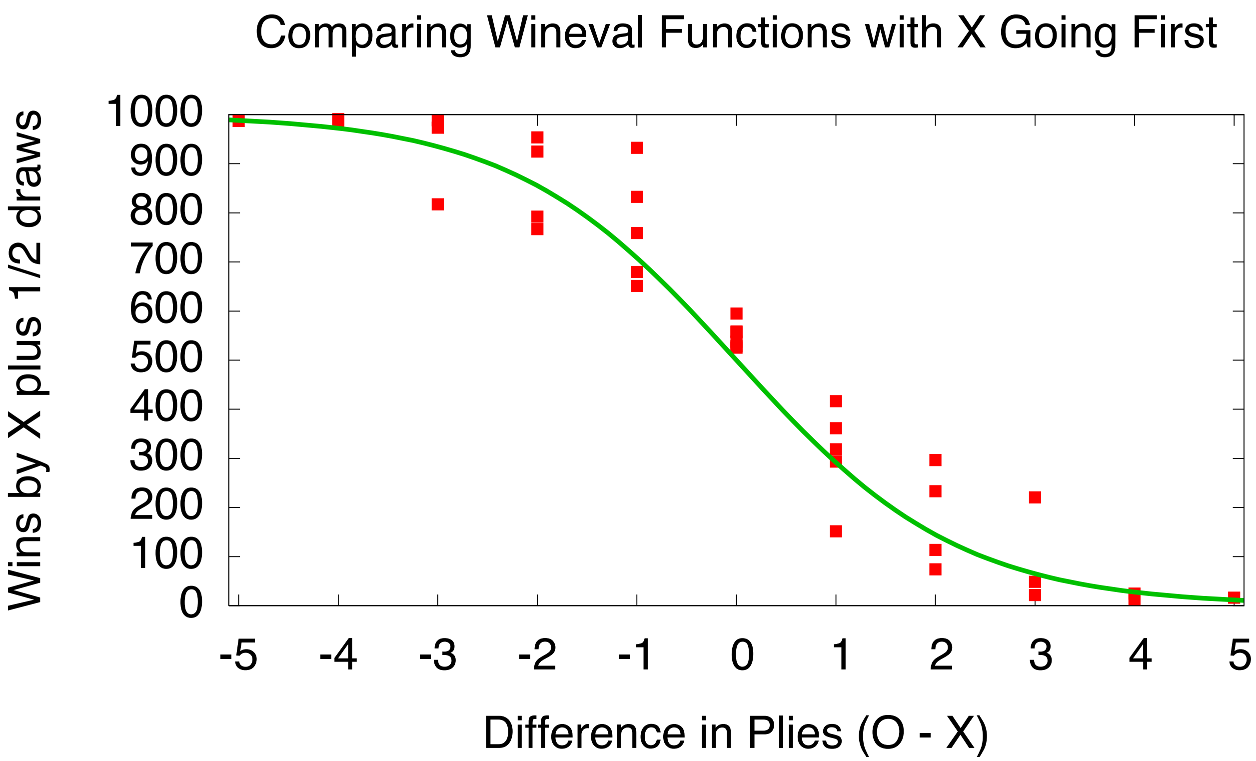

Figure 6: The first player has an advantage, but this advantage is not large enough to offset a difference in ply depth between players. The curve shown is a logistic function fitted to the data by gnuplot.

Fevala: Alternating First Move

The value shown is wins by the X player plus one-half of the draws from 1000 games, when X and O alternate who moves first.

O↓ X→	random	1ply	2ply	3ply	4ply	5ply
random	504.5					
1ply	27	497				
2ply	33	216	495			
3ply	5	208.5	370	517		
4ply	0	0	379.5	422	488.5	
5ply	2	406.5	290.5	521	650	482.5

Amazingly, 5ply consistently performs much worse than 3 or 4 ply, and plays worse against 1ply than 2ply. See Figure 7 for a graph.

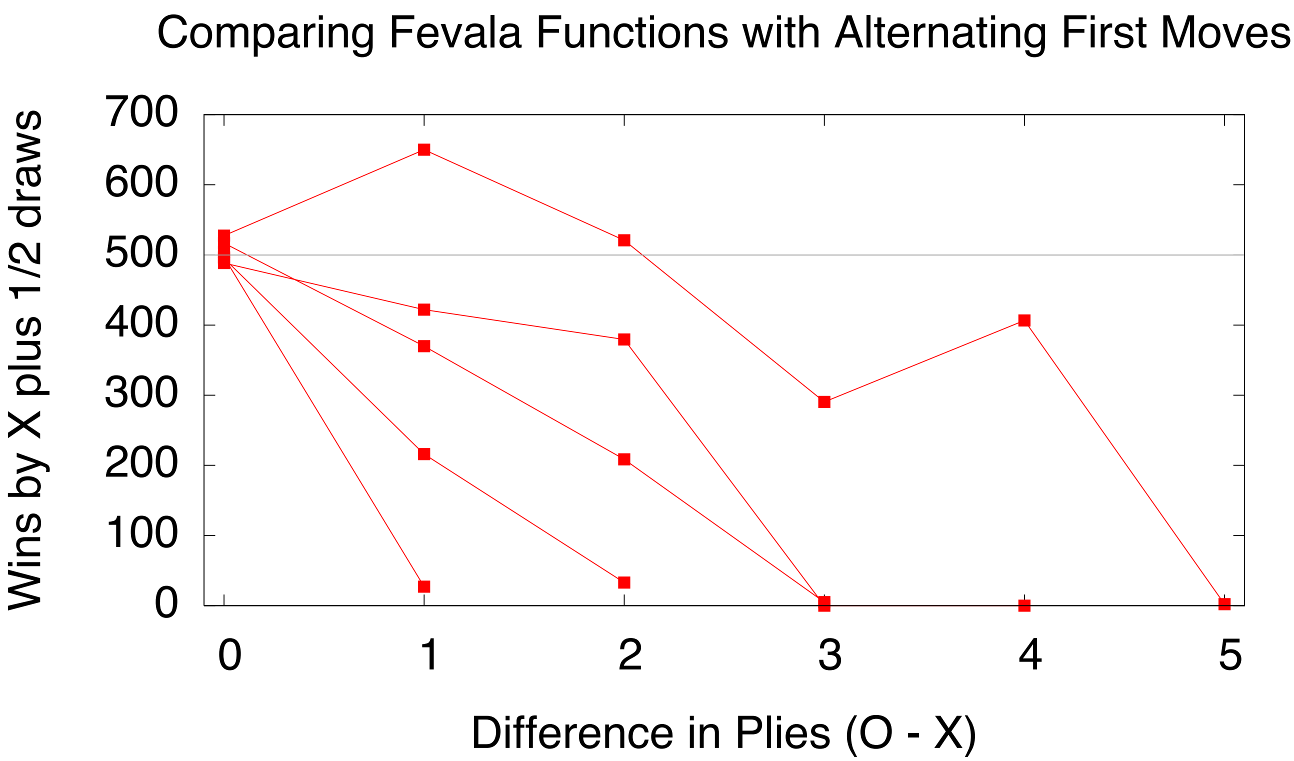

Figure 7: The top line is when the O player is at 5ply, the next line when it is at 4ply, and so on. Note that the 5ply player loses against both the 4ply and 3ply players.

Fevala: X Moves First

The value shown is wins by the X player plus one-half of the draws from 1000 games, when X always moves first. Figure 8 shows this data.

O↓ X→	**random**	**1ply**	**2ply**	**3ply**	**4ply**	**5ply**
random	532.5	998	987	1000	999	999
1ply	72	561	822	1000	1000	774
2ply	58	232	418.5	755.5	505.5	985
3ply	13	400.5	489.5	775	736.5	583.5
4ply	0	0	287	558.5	538	384.5
5ply	6	606.5	609.5	630	669	724

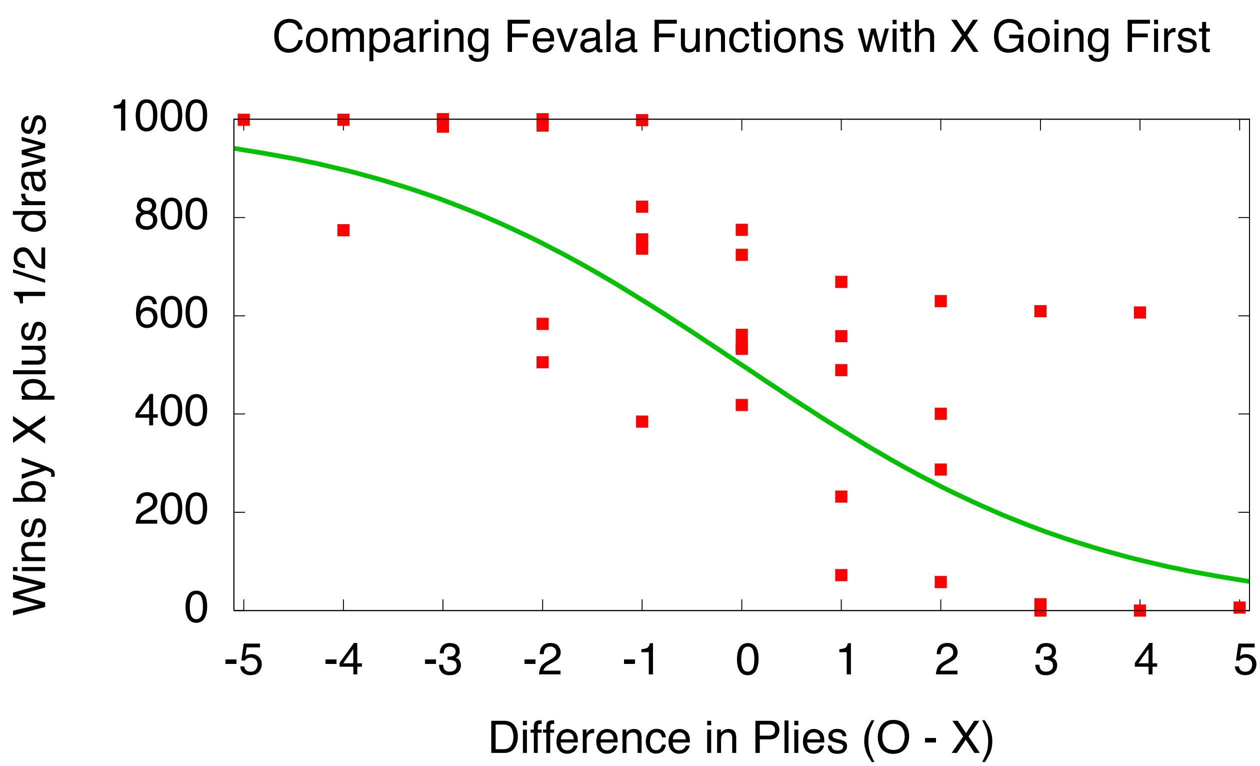

Figure 8: Going first gives fevala an advantage as it does wineval. The advantage is more notable at higher plies, and 2ply seems to be disadvantaged *by going first.*

Fevala vs. Wineval: Alternating First Move

The value shown is wins by the X player plus one-half of draws from 1000 games, when X (fevala) and O (wineval) alternate first move.

O↓ X→	**1ply**	**2ply**	**3ply**	**4ply**	**5ply**
1ply	898	933	993	1000	997
2ply	710	880.5	960.5	974	967.5
3ply	701.5	723	927.5	987.5	956
4ply	632.5	679	883	940.5	929
5ply	620.5	664.5	885	934.5	898

Fevala is superior to wineval considering win percentage, regardless of the number of plies for each player. Figure 9 is a graph of this data.

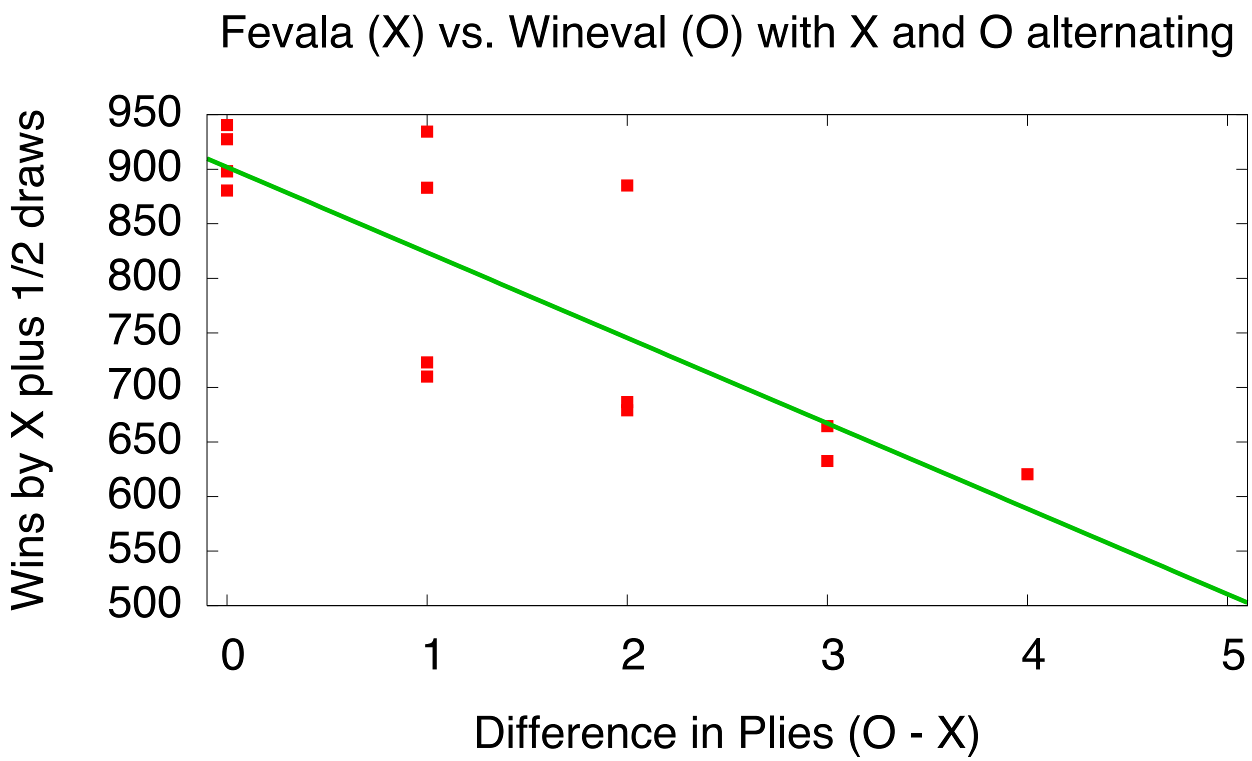

Figure 9: This graph shows that, when alternating first move, fevala plays better than wineval for a difference of less than five plies. The line shown is straight, because the logistic function no longer fits the data well.

Fevala vs. Wineval: X Moves First

This experiment has two parts. In the first, X is fevala and O wineval, in the second their evaluation functions are swapped.

O↓ X→	1ply fevala	2ply fevala	3ply fevala	4ply fevala	5ply fevala
1ply wineval	981.5	967	997	999	1000
2ply wineval	821	928	981	976.5	981.5
3ply wineval	809.5	840.5	968	980.5	979
4ply wineval	763.5	812.5	908.5	949.5	955
5ply wineval	732.5	809	902.5	951	951.5

O↓ X→	1ply wineval	2ply wineval	3ply wineval	4ply wineval	5ply wineval
1ply fevala	188	378	505	540	508.5
2ply fevala	108	181	417.5	468	451
3ply fevala	15	44.5	93.5	127.5	150.5
4ply fevala	1	28.5	26.5	63.5	86
5ply fevala	5	49.5	56.5	121	163

With the added advantage of first move, fevala merely becomes even more likely to win. If wineval is given the advantage of first move, it can play better than fevala for the 3ply, 4ply, and 5ply wineval against 1ply fevala. Figures 10 and 11 show this data.

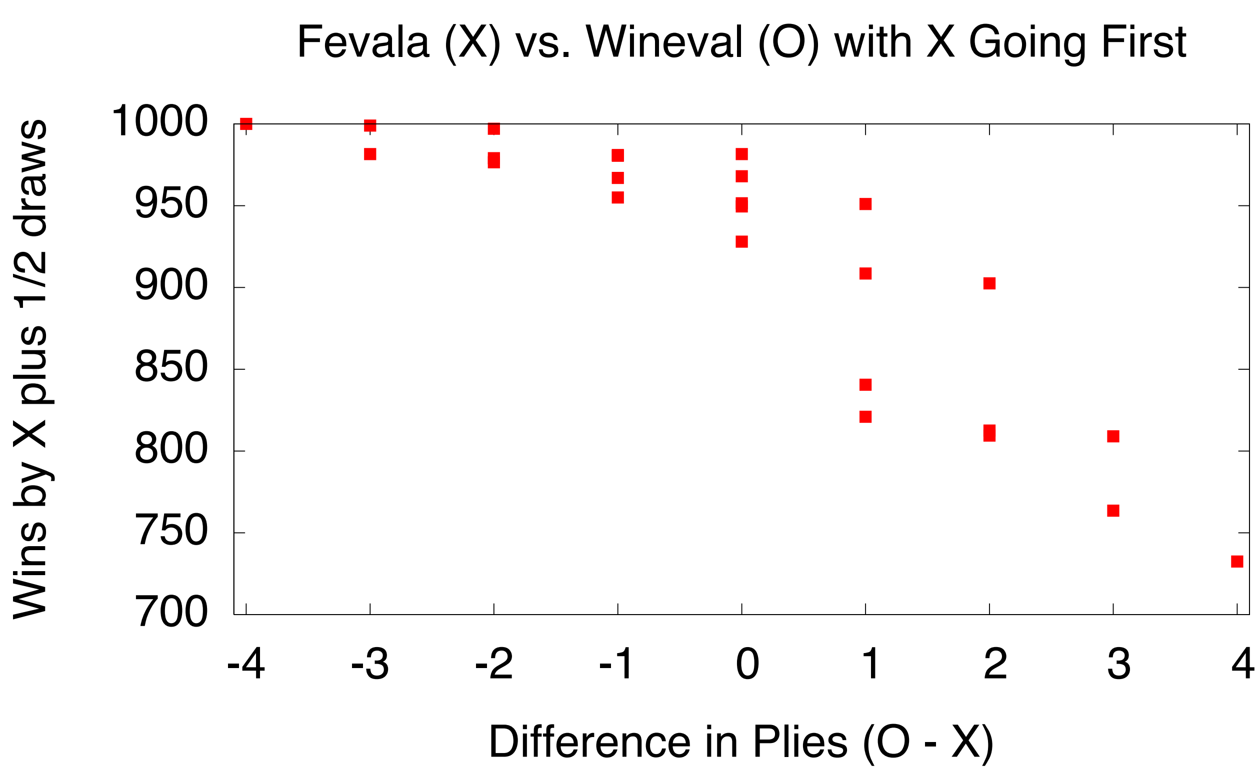

Figure 10: Fevala will beat wineval when wineval is less than 5 plies deeper and fevala goes first.

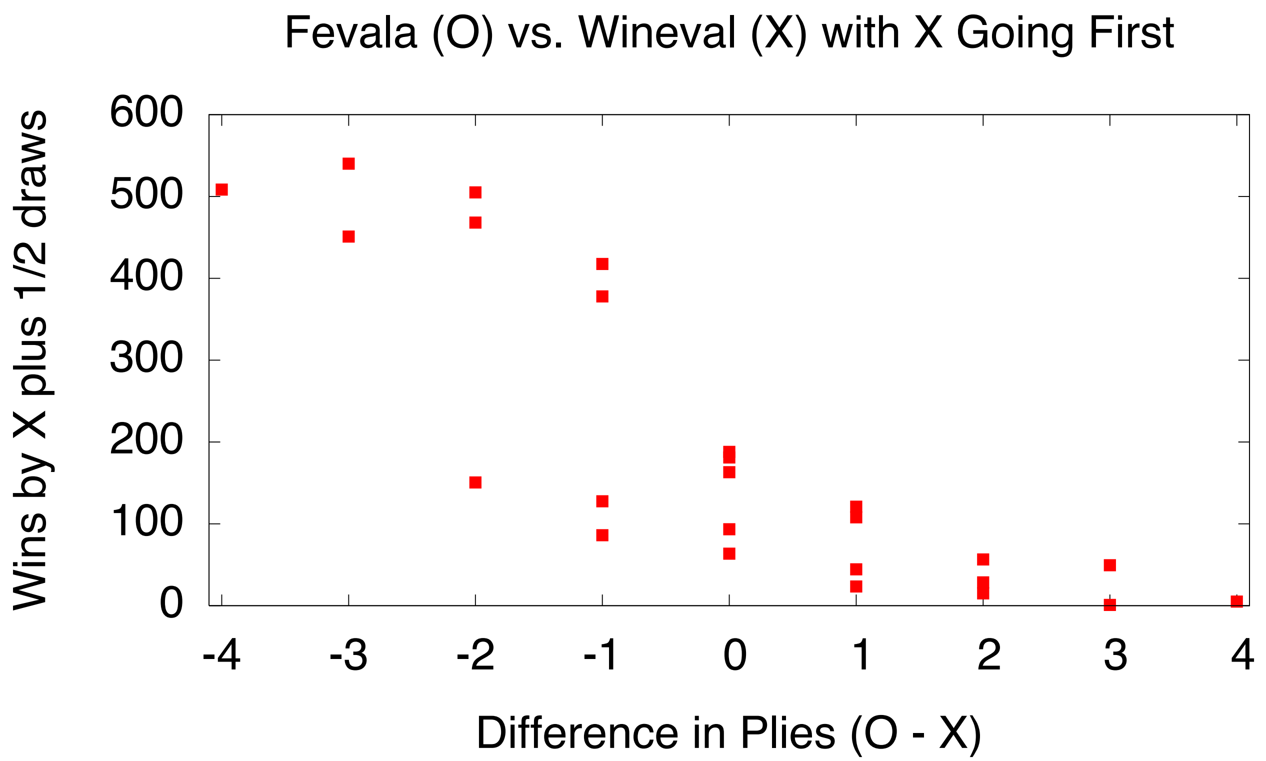

Figure 11: Wineval can beat fevala if it goes first and has a large advantage in ply depth.

Fevala, Fevalb, Fevalc, and Wineval

This is the only experiment which uses the fevalbv and fevalc functions. In this experiment, players are matched against others of the same ply depth only. In the table, A, B, and C represent the three feval functions while W stands for wineval. The value shown is wins by X plus one-half draws from 100 games.

X vs. O	B vs. C	B vs. A	A vs. C	B vs. W	W vs. C
1ply	19	62.5	36	98	4
2ply	25	78	66	94	6.5
3ply	42	54.5	87.5	97	13
4ply	89	65.5	25.5	96	5
5ply	80	75.5	44	98.5	4

At all ply depths, B and C can both beat W. At 1ply, the ordering is simple—C beats B beats A. At two and three plies, C beats B and B beats A, but A beats C. At four and five plies, the ordering is again simple—B beats C beats A.

Time Per Move

Here is the time per move in seconds that each player takes when faced against itself.

1ply wineval	2ply wineval	3ply wineval	4ply wineval	5ply wineval
0.000007	0.000052	0.000371	0.002097	0.015018

1ply fevala	2ply fevala	3ply fevala	4ply fevala	5ply fevala
0.000065	0.000430	0.002858	0.012625	0.129163

Note that each increase of one ply takes approximately seven times as long and that fevala takes about nine times as long as wineval. Figure 12 is a graph of these times.

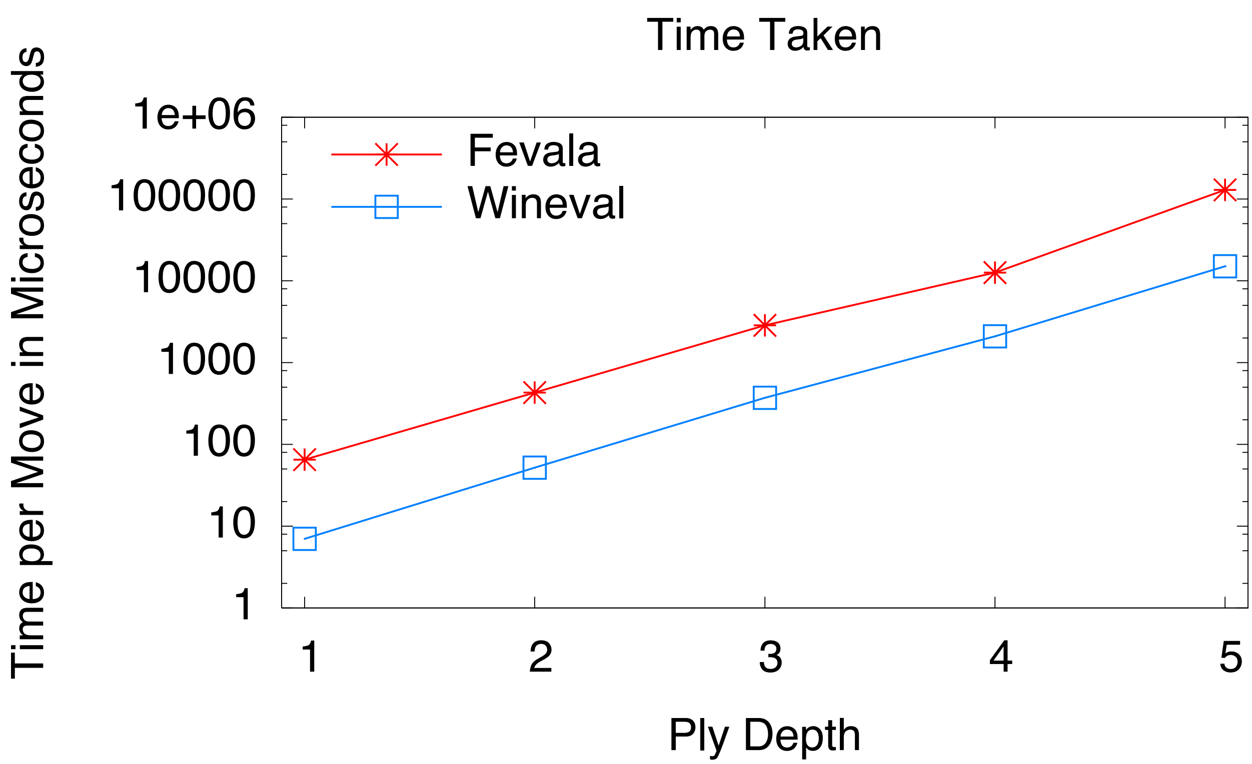

Figure 12: Time taken per move (displayed on a log scale).

Conclusions

My results showed that increasing ply depth increases the chance of winning, and that going first also increases the chance of winning, but usually not by as much as an increase in ply depth. This all agrees with my hypotheses. One result that I did not expect was the pattern of increased draws at even ply depths between identical wineval players.

Fevala plays much better than wineval, which is not what I expected. I hypothesized that it would only be slightly better. The fevalb and fevalc programs played stronger than fevala on average, though there were some interesting exceptions.

I noted that a 2-ply player performed much better than a 1-ply player without a noticeable decrease in speed (though theoretically, an extra ply takes 7 times as long). A 5-ply search took much longer than a 4-ply search, but played only slightly better. These diminishing returns are a common phenomenon with brute-force search programs. At some point improved performance is only feasible through smarter programs that do not search as much, such as the feval programs I wrote.

While doing this project, I improved my knowledge of C, learned some tcsh and some gnuplot, and learned some strategies for winning at Connect-Four. I also found out (again) how much work it takes to do a good science fair project.

Future Work

I plan to work on smarter evaluation functions. I have written three feval-type functions, and have one more idea for those. Using multiple regression would allow me to determine the “optimal” set of constants for a feval function. One other thing to consider is a more thorough evaluation function that looks at how many moves would be required to complete a four.

I also want to change the board representation, because it is space-inefficient, and a more compact version would allow me to implement a faster version of `invertboard` using table lookup. A possible alternative representation would change the columns-as-arrays model to columns-as-bytes, shrinking by a factor of 6. A column byte would begin with a number of zeros one more than the number of empty cells, followed by a one. The remaining bits would correspond to the filled cells of the column, with `1`s representing `'X'`s and `0`s, `'O'`s.

Acknowledgments

I wish to thank my father, Kevin Karplus, for mentoring me on this project. (See his Mentor Statement.) I also wish to thank my mother, Michele Hart, for her support and patience.

References

The information about a "Perfect Player" for Connect-Four came from

- The Wikipedia article on Connect-Four at
  `http://en.wikipedia.org/wiki/Connect_Four`
- "John's Connect Four Playground" at
  `http://homepages.cwi.nl/~tromp/c4/c4.html`

The min-max algorithm came from

- *Problem-Solving Methods in Artificial Intelligence*
  by Nils J. Nilson. McGraw-Hill, 1971.

Any other background information in this report I either already knew before beginning this project or learned from my father.

Mentor Statement

Abe started this project with some facility in C, but without much experience of recursive programming. He did a little reading on artificial intelligence and discussed with me how to structure his program. All the code is his own—I provided only minimal debugging help once or twice when he got stuck. We also designed together the more compact data structure (using only 7 bytes to represent the board), but he decided to delay implementing that.

I provided him some direct instruction on using tcsh scripts and gnumake to run his experiments, but almost all the scripting is his own. I also reminded him how to use gnuplot and showed him how to get it to produce PDF output.

For the experimental design, I suggested the experiments of comparing players with different numbers of plies and of determining how valuable the first move is for different players. The analysis of the results is his own. He also found on his own the result in the literature that a perfect first player has a forced win.

Appendix 1: Results

Experiment 1 (Wineval Alternating First Move)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
random	random	497	501	2	10414	10415	0.150523	0.000007
random	1ply_wineval	246	754	0	7900	8154	0.090368	0.000006
random	2ply_wineval	57	941	2	8697	9139	0.548404	0.000031
random	3ply_wineval	49	950	1	8576	9026	3.524387	0.000200
random	4ply_wineval	11	987	2	8344	8832	23.131893	0.001347
random	5ply_wineval	7	989	4	8326	8817	154.852191	0.009033
1ply_wineval	1ply_wineval	488	512	0	6951	6963	0.092912	0.000007
1ply_wineval	2ply_wineval	110	889	1	8840	9230	0.572790	0.000032
1ply_wineval	3ply_wineval	100	897	3	8096	8495	3.351376	0.000202
1ply_wineval	4ply_wineval	24	976	0	8083	8559	22.517733	0.001353
1ply_wineval	5ply_wineval	21	978	1	8178	8656	152.462117	0.009057
2ply_wineval	2ply_wineval	441	439	120	14962	14964	1.560349	0.000052
2ply_wineval	3ply_wineval	324	633	43	12429	12585	5.164620	0.000206
2ply_wineval	4ply_wineval	173	736	91	13164	13452	31.118808	0.001169
2ply_wineval	5ply_wineval	156	778	66	13107	13424	199.397897	0.007516
3ply_wineval	3ply_wineval	503	469	28	10964	10948	8.134313	0.000371
3ply_wineval	4ply_wineval	251	677	72	12958	13170	34.749831	0.001330
3ply_wineval	5ply_wineval	221	740	39	12044	12305	196.981023	0.008090
4ply_wineval	4ply_wineval	436	406	158	16491	16466	69.123174	0.002097
4ply_wineval	5ply_wineval	339	540	121	15120	15214	248.745051	0.008200
5ply_wineval	5ply_wineval	458	496	46	13741	13759	413.004694	0.015018

Experiment 2 (Wineval X Going First)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
random	random	556	442	2	10736	10180	0.162435	0.000008
random	1ply_wineval	318	681	1	8599	8281	0.094553	0.000006
random	2ply_wineval	74	926	0	9557	9483	0.573206	0.000030
random	3ply_wineval	48	951	1	9298	9250	3.613321	0.000195
random	4ply_wineval	12	987	1	9192	9180	23.926456	0.001302
random	5ply_wineval	15	982	3	9151	9136	159.121554	0.008701
1ply_wineval	random	832	167	1	8101	7269	0.086541	0.000006
1ply_wineval	1ply_wineval	595	405	0	7267	6672	0.093161	0.000007
1ply_wineval	2ply_wineval	150	847	3	9320	9170	0.576578	0.000031
1ply_wineval	3ply_wineval	112	885	3	9212	9100	3.570007	0.000195
1ply_wineval	4ply_wineval	21	978	1	9141	9120	24.010559	0.001315
1ply_wineval	5ply_wineval	23	974	3	9051	9028	157.104079	0.008690
2ply_wineval	random	952	45	3	8921	7969	0.530193	0.000031
2ply_wineval	1ply_wineval	932	67	1	8744	7812	0.538539	0.000033
2ply_wineval	2ply_wineval	483	432	85	14707	14224	1.522027	0.000053
2ply_wineval	3ply_wineval	331	608	61	12858	12527	5.151549	0.000203
2ply_wineval	4ply_wineval	192	726	82	14278	14086	32.166757	0.001134
2ply_wineval	5ply_wineval	189	748	63	13748	13559	199.136933	0.007293
3ply_wineval	random	973	26	1	8338	7365	3.280984	0.000209
3ply_wineval	1ply_wineval	924	74	2	8505	7581	3.341849	0.000208
3ply_wineval	2ply_wineval	654	295	51	12553	11899	5.161452	0.000211
3ply_wineval	3ply_wineval	550	433	17	11148	10598	8.103675	0.000373
3ply_wineval	4ply_wineval	259	672	69	13538	13279	35.350784	0.001318
3ply_wineval	5ply_wineval	267	674	59	13134	12867	199.691566	0.007680
4ply_wineval	random	991	9	0	8353	7362	22.013469	0.001401
4ply_wineval	1ply_wineval	988	11	1	8472	7484	22.286577	0.001397
4ply_wineval	2ply_wineval	752	167	81	13301	12549	31.108225	0.001203
4ply_wineval	3ply_wineval	731	213	56	12748	12017	34.036144	0.001374
4ply_wineval	4ply_wineval	449	374	177	16403	15954	68.418902	0.002115
4ply_wineval	5ply_wineval	368	535	97	15391	15023	243.861902	0.008018
5ply_wineval	random	984	10	6	8291	7307	148.054010	0.009492
5ply_wineval	1ply_wineval	987	12	1	8195	7208	145.736099	0.009462
5ply_wineval	2ply_wineval	785	150	65	13047	12262	197.304345	0.007796
5ply_wineval	3ply_wineval	739	205	56	12447	11708	198.035761	0.008199
5ply_wineval	4ply_wineval	607	305	88	15162	14555	252.897540	0.008510
5ply_wineval	5ply_wineval	499	442	59	14053	13554	415.972400	0.015068

Expermiment 3 (Fevala Alternating First Move)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
random	random	503	494	3	10420	10414	0.091903	0.000004
random	1ply_fevala	27	973	0	5175	5648	0.479670	0.000044
random	2ply_fevala	33	967	0	7017	7484	4.774327	0.000329
random	3ply_fevala	5	995	0	6246	6741	29.797872	0.002294
random	4ply_fevala	0	1000	0	5959	6459	192.480956	0.015500
1ply_fevala	1ply_fevala	121	127	752	20096	20112	2.617723	0.000065
1ply_fevala	2ply_fevala	93	661	246	16523	16801	8.493568	0.000255
1ply_fevala	3ply_fevala	130	713	157	10749	11119	31.460775	0.001439
1ply_fevala	4ply_fevala	0	1000	0	6000	6500	163.879137	0.013110
2ply_fevala	2ply_fevala	371	381	248	18632	18637	16.014545	0.000430
2ply_fevala	3ply_fevala	272	532	196	13769	13882	44.115633	0.001595
2ply_fevala	4ply_fevala	207	448	345	19081	19207	269.283029	0.007033
3ply_fevala	3ply_fevala	419	385	196	13629	13613	77.851504	0.002858
3ply_fevala	4ply_fevala	280	436	284	16717	16887	310.891661	0.009252
4ply_fevala	4ply_fevala	175	198	627	20332	20347	513.563980	0.012625
5ply_fevala	random	998	2	0	7112	6614	1451.252625	0.105730
5ply_fevala	1ply_fevala	538	351	111	13512	13382	1465.047087	0.054475
5ply_fevala	2ply_fevala	692	273	35	10316	10090	1367.460327	0.067013
5ply_fevala	3ply_fevala	446	488	66	14664	14686	1770.603759	0.060327
5ply_fevala	4ply_fevala	263	563	174	17007	17161	2193.715658	0.064204
5ply_fevala	5ply_fevala	446	501	53	14699	14725	3800.487834	0.129163

Experiment 4 (Fevala X Moves First)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
random	random	531	466	3	10688	10157	0.183748	0.000009
random	1ply_fevala	72	928	0	5606	5534	0.486893	0.000044
random	2ply_fevala	58	942	0	7543	7485	4.780730	0.000318
random	3ply_fevala	3	987	0	6790	6777	30.278717	0.002232
random	4ply_fevala	0	1000	0	6933	6933	205.160340	0.014796
random	5ply_fevala	6	994	0	7527	7521	1529.164281	0.101619
1ply_fevala	random	998	2	0	5639	4641	0.468916	0.000046
1ply_fevala	1ply_fevala	187	65	748	20179	19992	2.621129	0.000065
1ply_fevala	2ply_fevala	120	656	224	17632	17512	8.411947	0.000239
1ply_fevala	3ply_fevala	259	458	283	16348	16089	40.056500	0.001235
1ply_fevala	4ply_fevala	0	1000	0	7000	7000	177.062922	0.012647
1ply_fevala	5ply_fevala	508	295	197	16433	15925	1496.392133	0.046245
2ply_fevala	random	987	13	0	7757	6770	4.922662	0.000339
2ply_fevala	1ply_fevala	701	57	242	16333	15632	8.632281	0.000270
2ply_fevala	2ply_fevala	291	454	255	18747	18456	16.063683	0.000432
2ply_fevala	3ply_fevala	410	431	159	14251	13841	42.522238	0.001514
2ply_fevala	4ply_fevala	111	537	352	18793	18682	266.514367	0.007112
2ply_fevala	5ply_fevala	578	359	63	16060	15482	1757.620079	0.055723
3ply_fevala	random	1000	0	0	6328	5328	27.983688	0.002401
3ply_fevala	1ply_fevala	1000	0	0	6000	5000	23.285308	0.002117
3ply_fevala	2ply_fevala	645	134	221	13540	12895	45.257284	0.001712
3ply_fevala	3ply_fevala	680	130	190	14517	13837	79.617377	0.002808
3ply_fevala	4ply_fevala	296	179	525	19630	19334	317.196270	0.008141
3ply_fevala	5ply_fevala	580	320	100	16278	15698	1716.850645	0.053692
4ply_fevala	random	999	1	0	6234	5235	185.184713	0.016147
4ply_fevala	1ply_fevala	1000	0	0	6000	5000	150.951882	0.013723
4ply_fevala	2ply_fevala	349	338	313	19831	19482	275.243724	0.007001
4ply_fevala	3ply_fevala	671	198	131	15034	14363	305.810867	0.010403
4ply_fevala	4ply_fevala	244	168	588	20419	20175	515.996809	0.012711
4ply_fevala	5ply_fevala	596	258	146	16739	16143	2099.689702	0.063855
5ply_fevala	random	999	1	0	6317	5318	1310.094828	0.112599
5ply_fevala	1ply_fevala	752	204	44	10980	10228	1438.959035	0.067850
5ply_fevala	2ply_fevala	984	14	2	5104	4120	1003.086850	0.108747
5ply_fevala	3ply_fevala	560	393	47	13578	13018	1799.486138	0.067660
5ply_fevala	4ply_fevala	311	542	147	17564	17253	2241.673507	0.064384
5ply_fevala	5ply_fevala	694	246	60	15108	14414	3789.287762	0.128355

Experiment 5 (Fevala vs. Wineval Altenating First Move)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
1ply_fevala	1ply_wineval	898	102	0	5553	5155	0.519663	0.000049
1ply_fevala	2ply_wineval	700	280	20	12178	11968	1.504575	0.000062
1ply_fevala	3ply_wineval	672	299	29	11881	11696	4.413046	0.000187
1ply_fevala	4ply_wineval	610	345	45	13126	12993	24.035188	0.000920
1ply_fevala	5ply_wineval	602	361	37	12833	12715	140.313729	0.005492
2ply_fevala	1ply_wineval	933	67	0	7449	7016	4.789170	0.000331
2ply_fevala	2ply_wineval	873	112	15	14297	13915	8.174624	0.000290
2ply_fevala	3ply_wineval	717	271	12	12260	12036	10.108384	0.000416
2ply_fevala	4ply_wineval	669	311	20	13782	13601	30.912409	0.001129
2ply_fevala	5ply_wineval	653	324	23	13506	13339	154.002724	0.005737
3ply_fevala	1ply_wineval	993	7	0	6495	6002	28.864545	0.002310
3ply_fevala	2ply_wineval	958	37	5	11718	11257	43.698275	0.001902
3ply_fevala	3ply_wineval	924	69	7	11615	11187	46.283537	0.002030
3ply_fevala	4ply_wineval	876	110	14	13529	13146	72.046512	0.002701
3ply_fevala	5ply_wineval	877	107	16	13675	13290	203.487120	0.007546
4ply_fevala	1ply_wineval	1000	0	0	6649	6149	196.976401	0.015391
4ply_fevala	2ply_wineval	968	20	12	12016	11539	279.237152	0.011855
4ply_fevala	3ply_wineval	983	8	9	12210	11722	284.669361	0.011895
4ply_fevala	4ply_wineval	927	46	27	14672	14232	336.883562	0.011655
4ply_fevala	5ply_wineval	925	56	19	14781	14343	475.875599	0.016340
5ply_fevala	1ply_wineval	997	3	0	6864	6367	1420.274291	0.107344
5ply_fevala	2ply_wineval	962	27	11	10717	10247	1802.350628	0.085974
5ply_fevala	3ply_wineval	950	38	12	10651	10193	1801.209466	0.086414
5ply_fevala	4ply_wineval	919	61	20	13132	12703	2086.836174	0.080776
5ply_fevala	5ply_wineval	893	97	10	13087	12686	2233.220027	0.086650

Experiment 6 (Fevala vs. Wineval X Moves First)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
1ply_fevala	1ply_wineval	981	18	1	5595	4614	0.473865	0.000046
1ply_fevala	2ply_wineval	807	165	28	13475	12668	1.613418	0.000062
1ply_fevala	3ply_wineval	794	175	31	13645	12851	4.732477	0.000179
1ply_fevala	4ply_wineval	743	216	41	14853	14110	25.356432	0.000875
1ply_fevala	5ply_wineval	707	242	51	14835	14128	147.967513	0.005109
2ply_fevala	1ply_wineval	967	33	0	7593	6626	4.826172	0.000339
2ply_fevala	2ply_wineval	926	70	4	15012	14086	8.451764	0.000290
2ply_fevala	3ply_wineval	835	154	11	14053	13218	11.133153	0.000408
2ply_fevala	4ply_wineval	805	180	15	15701	14896	32.959816	0.001077
2ply_fevala	5ply_wineval	798	180	22	15691	14893	161.239978	0.005272
3ply_fevala	1ply_wineval	997	3	0	6398	5401	28.081140	0.002380
3ply_fevala	2ply_wineval	979	17	4	11346	10367	42.604064	0.001962
3ply_fevala	3ply_wineval	966	30	4	11397	10431	45.580101	0.002088
3ply_fevala	4ply_wineval	902	85	13	13696	12794	71.827199	0.002711
3ply_fevala	5ply_wineval	897	92	11	13750	12853	196.837921	0.007399
4ply_fevala	1ply_wineval	999	1	0	6255	5256	184.988648	0.016071
4ply_fevala	2ply_wineval	970	17	13	11501	10531	270.337963	0.012270
4ply_fevala	3ply_wineval	976	15	9	11690	10714	275.406297	0.012293
4ply_fevala	4ply_wineval	938	39	23	14922	13984	339.886666	0.011758
4ply_fevala	5ply_wineval	942	40	18	14658	13716	466.423895	0.016438
5ply_fevala	1ply_wineval	1000	0	0	6369	5369	1313.646757	0.111914
5ply_fevala	2ply_wineval	978	15	7	9341	8363	1627.818523	0.091946
5ply_fevala	3ply_wineval	976	18	6	9324	8348	1642.075407	0.092920
5ply_fevala	4ply_wineval	949	39	12	12717	11768	2053.958257	0.083886
5ply_fevala	5ply_wineval	948	45	7	12821	11873	2192.954572	0.088805

Continued...

Experiment 6 (Continued)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
1ply_wineval	1ply_fevala	188	812	0	5433	5245	0.457042	0.000043
2ply_wineval	1ply_fevala	365	609	26	11872	11507	1.446480	0.000062
3ply_wineval	1ply_fevala	494	484	22	10365	9871	3.964499	0.000196
4ply_wineval	1ply_fevala	522	442	36	11313	10791	22.022440	0.000996
5ply_wineval	1ply_fevala	492	475	33	11744	11252	139.311643	0.006058
1ply_wineval	2ply_fevala	108	892	0	7315	7207	4.623464	0.000318
2ply_wineval	2ply_fevala	172	810	18	13758	13586	7.907085	0.000289
3ply_wineval	2ply_fevala	408	573	19	11278	10870	9.302468	0.000420
4ply_wineval	2ply_fevala	460	524	16	11961	11501	28.453092	0.001213
5ply_wineval	2ply_fevala	439	537	24	12209	11770	149.288671	0.006226
1ply_wineval	3ply_fevala	15	985	0	6993	6978	31.003620	0.002219
2ply_wineval	3ply_fevala	43	954	3	12083	12040	45.246221	0.001876
3ply_wineval	3ply_fevala	88	901	11	11665	11577	46.852030	0.002016
4ply_wineval	3ply_fevala	119	864	17	13476	13357	72.420128	0.002699
5ply_wineval	3ply_fevala	144	843	13	13355	13211	208.246130	0.007839
1ply_wineval	4ply_fevala	1	999	0	7073	7072	208.417507	0.014734
2ply_wineval	4ply_fevala	24	967	9	12759	12735	293.623474	0.011517
3ply_wineval	4ply_fevala	17	970	13	12575	12558	296.847063	0.011811
4ply_wineval	4ply_fevala	51	924	25	14897	14846	339.688916	0.011421
5ply_wineval	4ply_fevala	70	898	32	14780	14710	481.175829	0.016317
1ply_wineval	5ply_fevala	5	995	0	7492	7487	1528.014254	0.102010
2ply_wineval	5ply_fevala	45	946	9	11764	11719	1950.094710	0.083043
3ply_wineval	5ply_fevala	48	935	17	11589	11541	1933.460908	0.083591
4ply_wineval	5ply_fevala	112	870	18	13668	13556	2127.088871	0.078133
5ply_wineval	5ply_fevala	156	830	14	13559	13403	2274.412566	0.084356

Experiment 7 (Fevalb & Fevalc Alternating First Move)

Xname	Oname	Xwin	Owin	Draw	Xmoves	Omoves	GameTime	MoveTime
1ply_fevalb	1ply_fevalc	19	81	0	936	967	0.168212	0.000088
1ply_fevalb	1ply_fevala	54	29	17	1194	1173	0.165193	0.000070
1ply_fevala	1ply_fevalc	36	64	0	714	728	0.116501	0.000081
1ply_fevalb	1ply_wineval	98	2	0	576	528	0.050918	0.000046
1ply_wineval	1ply_fevalc	4	96	0	481	527	0.046651	0.000046
2ply_fevalb	2ply_fevalc	0	50	50	2000	2050	1.643705	0.000406
2ply_fevalb	2ply_fevala	75	19	6	1624	1597	1.383097	0.000429
2ply_fevala	2ply_fevalc	58	26	16	1765	1748	1.510294	0.000430
2ply_fevalb	2ply_wineval	94	6	0	1215	1171	0.729980	0.000306
2ply_wineval	2ply_fevalc	6	93	1	1235	1279	0.776053	0.000309
3ply_fevalb	3ply_fevalc	42	58	0	1531	1539	11.734061	0.003822
3ply_fevalb	3ply_fevala	41	32	27	1614	1612	9.304319	0.002884
3ply_fevala	3ply_fevalc	84	9	7	1523	1482	9.097150	0.003027
3ply_fevalb	3ply_wineval	96	2	2	1016	970	4.409285	0.002220
3ply_wineval	3ply_fevalc	10	84	6	1033	1070	4.724047	0.002246
4ply_fevalb	4ply_fevalc	89	11	0	1723	1684	68.813289	0.020198
4ply_fevalb	4ply_fevala	65	34	1	1808	1792	59.816335	0.016616
4ply_fevala	4ply_fevalc	24	73	3	1577	1603	67.019092	0.021075
4ply_fevalb	4ply_wineval	96	4	0	1302	1256	35.109534	0.013725
4ply_wineval	4ply_fevalc	3	93	4	1337	1380	36.811849	0.013549
5ply_fevalb	5ply_fevalc	76	16	8	1659	1633	421.632898	0.128078
5ply_fevalb	5ply_fevala	67	16	17	1570	1541	395.064745	0.126990
5ply_fevala	5ply_fevalc	41	53	6	1551	1557	405.253265	0.130390
5ply_fevalb	5ply_wineval	98	1	1	1206	1158	216.246834	0.091475
5ply_wineval	5ply_fevalc	3	95	2	1158	1203	222.007194	0.094031

Appendix 2: Makefile

Main program

```
connectfour:   boardcontrols.o connectfour-ncurses.o p-human.o
randperm.o recurse-player.o p-random.o p-wineval.o p-fevala.o
     gcc -g -o $@ $^ -lncurses

%.o: %.c
     gcc -g -c $^

%.pdf:    %.gplot
     gnuplot $^ > $@
```

Ncurses

```
board-disp: board-disp.o ../p-random.o ../p-wineval.o ../
boardcontrols.o ../recurse-player.o ../randperm.o
     gcc -o $@ $^ -lncurses
```

Appendix 3: tcsh

Experiment 1 Runner

```
#!/bin/tcsh

foreach o (random 1ply_wineval 2ply_wineval 3ply_wineval 4ply_wineval
5ply_wineval)
     connectfour -n 1000 -a -d -X random -O $o -t
end

foreach o (1ply_wineval 2ply_wineval 3ply_wineval 4ply_wineval 5ply_wineval)
     connectfour -n 1000 -a -d -X 1ply_wineval -O $o -t
end

foreach o (2ply_wineval 3ply_wineval 4ply_wineval 5ply_wineval)
     connectfour -n 1000 -a -d -X 2ply_wineval -O $o -t
end

foreach o (3ply_wineval 4ply_wineval 5ply_wineval)
     connectfour -n 1000 -a -d -X 3ply_wineval -O $o -t
end

foreach o (4ply_wineval 5ply_wineval)
     connectfour -n 1000 -a -d -X 4ply_wineval -O $o -t
end

foreach o (5ply_wineval)
     connectfour -n 1000 -a -d -X 5ply_wineval -O $o -t
end
```

Appendix 4: gnuplot

Experiment 1

```
set terminal pdf fsize 12
set xrange [-0.1:5.1]
set ylabel "Wins by X plus 1/2 draws"
set xlabel "Difference in Plies"
set title "Comparing Wineval Functions with Alternating First Moves"
unset key
f(x,a)=1000*exp(a*x)/(1+exp(a*x))
fit f(x,a) 'exp1-run2-fgp.txt' using ($5-$2):($12+($16*.5)) via a
                                  # ply_diff:xwin+.5*draw
plot 'exp1-run2-fgp.txt' using ($5-$2):($12+($16*.5)) w p pt 5, f(x,a) w l lw 5
```