Erika Sklaver, Amanda Milloy

Professor Majercik

Artificial Intelligence

September 26th, 2016

Minimax Search for Connect-4

**1 Connect-4 Description**

Connect-4 is a two-player game in which each player takes turns making a move. A move is made by dropping a disk onto a two-dimensional grid consisting (typically) of six rows and seven columns. A player wins by placing four of their own disks in a row, either horizontally, vertically, or diagonally.

**2 Minimax**

In a minimax search approach, the computer considers all of the possible actions of the game player and itself in order to determine the best possible move for a given turn.  It searches through these moves using a search tree. The “max” player moves first, so it is the root of the search tree and it has a finite number of moves to choose from for the first turn. In order to choose which move to make, max will look at each of their potential moves and each of the potential following moves that “min” could make. “Max” will then look at every potential move it could make on its second turn, given all the potential moves that min could have made. This process will continue until “max” has searched the entire tree and determined which would be the best possible move.

To implement the minimax algorithm, we give numeric values to each state (each move) and use recursion to find those values of each successor state.  In other words, we have functions “expandMaxNode” and “expandMinNode” which call each other and each return a minimax value. The recursion will go all the way down the tree and then minimax values are returned as the recursion unfolds.  So, the “max” nodes are trying to find a value that is greater than the current maximum value while the “min” nodes are trying to find a value less than the current minimum value.  When “max” decides which move to make, it will choose the move that leads to the path with the largest value and “min” will choose the opposite.

**2.1 Alpha-Beta Pruning**

The minimax search without pruning is a depth-first search of the search tree with time complexity O(b^m) and space complexity O(bm) (where b is the number of legal moves at each state and m is the depth of the tree).  This is impractical as it will take a long time and use up a lot of space. Alpha-beta pruning improves the time and space problems by having the computer ignore the parts of the search tree that will not make a difference in its choice of move for that turn. For example, if we are at node x, but the player has a better move to make further up in the tree, then there is no point in even looking at node x or any of its leaves because node x will never be reached when the game is actually played. So, we use alpha to store the highest value choice for max and beta to store the lowest value choice for min.

If we are at a max node and v is the value of the child just explored, if v<=alpha then “max” will keep searching in the tree for a better (higher value) move. If v is greater than alpha and less than beta, then we change alpha to be v, but still keep searching for a better move. And if v>= beta then we return v because, although this is the greatest value, we also know that there is a path that min will choose with a lower value, so min will never let max take this path. This is where some of the branches are “pruned.” Similarly, at a min node, if v>= beta then “min” will keep searching the tree for a better (lower value) move. If v is greater than alpha but less than beta, then we change beta to v, but still keep searching for a better move. And if v<=alpha then we return v because, although this is the lowest value, we know that there is a path that “max” can choose with a higher value, so max will never let “min” take this path. With this pruning, we are not searching through parts of the tree that will never be used in a real game, so this will improve the efficiency of the minimax search.

**3 Experimental Methodology**

For minimax search with alpha-beta pruning and minimax search without alpha-beta pruning, we recorded the total number of moves explored and the total exploration time for ten games on each of the eight levels. By averaging these numbers, we hoped to identify performance differences between the two methods of search.

**3.1 Data**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 | Level 6 | Level 7 | Level 8 |
| Run 1 | 0 | 0 | 1 | 15 | 97 | 259 | 3797 | 16648 |
| Run 2 | 0 | 0 | 1 | 34 | 82 | 549 | 2370 | 21612 |
| Run 3 | 0 | 0 | 1 | 25 | 78 | 240 | 2074 | 38471 |
| Run 4 | 0 | 0 | 0 | 10 | 47 | 318 | 1826 | 20049 |
| Run 5 | 0 | 0 | 1 | 5 | 67 | 443 | 3135 | 15924 |
| Run 6 | 0 | 0 | 1 | 19 | 67 | 268 | 4394 | 19528 |
| Run 7 | 0 | 0 | 3 | 14 | 94 | 404 | 3615 | 33490 |
| Run 8 | 0 | 0 | 1 | 8 | 99 | 337 | 3249 | 15083 |
| Run 9 | 0 | 0 | 1 | 14 | 76 | 451 | 2340 | 37947 |
| Run 10 | 0 | 0 | 0 | 7 | 57 | 438 | 2360 | 28986 |
| Average (ms) | 0 | 0 | 1 | 15.1 | 76.4 | 370.7 | 2916 | 24773.8 |

**Figure 1. The total time in milliseconds for ten games at each level without alpha-beta pruning.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 | Level 6 | Level 7 | Level 8 |
| Run 1 | 0 | 0 | 1 | 7 | 17 | 56 | 137 | 867 |
| Run 2 | 0 | 0 | 1 | 6 | 18 | 52 | 147 | 420 |
| Run 3 | 0 | 0 | 1 | 6 | 15 | 69 | 209 | 368 |
| Run 4 | 0 | 1 | 2 | 9 | 16 | 67 | 72 | 362 |
| Run 5 | 0 | 0 | 2 | 7 | 19 | 30 | 97 | 992 |
| Run 6 | 0 | 0 | 1 | 6 | 19 | 68 | 153 | 396 |
| Run 7 | 0 | 1 | 2 | 8 | 22 | 69 | 148 | 888 |
| Run 8 | 0 | 1 | 2 | 10 | 15 | 65 | 70 | 984 |
| Run 9 | 0 | 1 | 1 | 6 | 13 | 58 | 196 | 393 |
| Run 10 | 0 | 0 | 1 | 8 | 23 | 51 | 150 | 914 |
| Average (ms) | 0 | 0.4 | 1.4 | 7.3 | 17.7 | 58.5 | 137.9 | 658.4 |

**Figure 2. The total time in milliseconds for ten games at each level with alpha-beta pruning.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 | Level 6 | Level 7 | Level 8 |
| Run 1 | 24 | 568 | 1640 | 47546 | 313360 | 1289998 | 17304312 | 86262051 |
| Run 2 | 80 | 424 | 5064 | 39794 | 334187 | 2547754 | 10492409 | 118057190 |
| Run 3 | 24 | 853 | 6586 | 33681 | 310926 | 1198501 | 9716963 | 159833348 |
| Run 4 | 56 | 488 | 3098 | 27023 | 170495 | 1625126 | 8483006 | 90855722 |
| Run 5 | 112 | 780 | 3373 | 17520 | 277859 | 2092271 | 12691535 | 82488393 |
| Run 6 | 40 | 329 | 4420 | 56406 | 318604 | 1392588 | 20595298 | 101433974 |
| Run 7 | 88 | 424 | 5704 | 48760 | 355632 | 2131635 | 15207229 | 134787896 |
| Run 8 | 56 | 706 | 5004 | 35160 | 367889 | 1595992 | 13696392 | 88284658 |
| Run 9 | 24 | 798 | 2264 | 41792 | 263334 | 2089318 | 9812083 | 127748469 |
| Run 10 | 72 | 631 | 5231 | 26485 | 221441 | 2399056 | 10973347 | 93246657 |
| Average | 57.6 | 600.1 | 4238.4 | 37416.7 | 293372.7 | 1836223.9 | 12897257.4 | 108299835.8 |

**Figure 3. The total number of moves explored for ten games at each level without alpha-beta pruning.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 | Level 6 | Level 7 | Level 8 |
| Run 1 | 32 | 237 | 953 | 7276 | 57257 | 265117 | 735548 | 5340900 |
| Run 2 | 32 | 264 | 2053 | 8021 | 64300 | 247388 | 838410 | 2520699 |
| Run 3 | 32 | 492 | 1322 | 10595 | 35273 | 353512 | 1231327 | 2274704 |
| Run 4 | 48 | 372 | 1686 | 15132 | 47388 | 323735 | 399141 | 2146371 |
| Run 5 | 48 | 295 | 1767 | 13982 | 57257 | 114934 | 570829 | 5871925 |
| Run 6 | 24 | 250 | 953 | 8402 | 64370 | 341276 | 972848 | 2268463 |
| Run 7 | 40 | 239 | 1881 | 11801 | 81413 | 323735 | 831549 | 5340900 |
| Run 8 | 24 | 214 | 953 | 13197 | 49532 | 296853 | 309964 | 5889210 |
| Run 9 | 32 | 292 | 1087 | 6742 | 34646 | 261790 | 1153227 | 2146371 |
| Run 10 | 48 | 319 | 1056 | 14180 | 80178 | 252123 | 838410 | 5362717 |
| Average | 36 | 297.4 | 1371.1 | 10932.8 | 57161.4 | 278046.3 | 788125.3 | 3916226 |

**Figure 4. The total number of moves explored for ten games at each level with alpha-beta pruning.**

**Figure 5. A comparison between the total time in milliseconds for ten games at each level with alpha-beta pruning and with no alpha-beta pruning using a logarithmic scale.**

**Figure 6. A comparison between total moves explored for ten games at each level with alpha-beta pruning and with no alpha-beta pruning using a logarithmic scale.**

**4 Results**

We found that the minimax search performed much better with alpha-beta pruning than without. In other words, our algorithm explored far fewer moves and resulted in a smaller total time with the use of pruning. At the highest level, the search without pruning explored over 100,000,000 more moves on average and took over 20,000 more milliseconds to complete. Our data showed that the proportional disparity between the two performances grew larger and larger as the levels increased.

**5 Discussion Questions**

**a) Explain the heuristic function. What quantities does it compute and how does it weight them?**

If a move could lead to four disks in a row, the individual heuristic returns a super large number (MAX\_VALUE) as that means the player would win the game. If, instead, the move would lead to the opponent player placing four disks in a row, the heuristic returns a very low value as that means the player would lose the game. If neither of these cases is true, the heuristic uses the number of three disks in a row, two disks in a row, and one disk in a row for each player to determine their respective strengths. Three disks in a row are weighted higher (multiplied by 32) than two disks in a row (multiplied by four). Both are weighted higher than a solo disk, which is not multiplied by anything. All three of these values are added up and returned as the individual strength. However, to calculate the full heuristic, the stats class returns the current player’s strength minus the other player’s strength, as the best move should lead to an overall positive output.

**b) Moves are always explored from left to right. There is no reason why they couldn’t be explored in some other order. What ordering do you think might be better here? Could to ordering matter in minimax without alpha-beta pruning? With alpha-beta pruning? In each case, explain what difference it could make, or why it couldn’t make a difference.**

Ordering in minimax without alpha-beta pruning will not make a difference in efficiency because we will still be searching through every single possibility, just in a different order. However, the ordering could make a difference in minimax with alpha-beta pruning. If we search through the worst successors first and the best successors last, then we will end up searching through the whole tree, as if alpha-beta pruning were not implemented. For example, if we are at a max node, and v is the value of the successor just explored, if v<= alpha then we will keep searching. The pruning happens when we reach a v>=beta, because we know that this is the highest possible value “max” could get, so we can stop searching at this point.  So if we start with the worst successors, nothing will be “pruned.” However, if we could order the search so that we explore the best successors first, then we will immediately get a value v>=beta, so we can prune a lot of branches.

**c) This version of Connect-4 has 8 columns, while most have 7 columns. This program can be changed to a 7-column version by changing the constant NUMBER\_OF\_COLUMNS to 7 and by changing the unnamed constant 84 on line 97 of that file to another value. What should that value be and why? Wouldn’t it be nice if it were a named constant?**

If one column is eliminated from the board, the unnamed constant should be changed to 69. This value represents the number of valid combinations of four in a row on the board. By taking away a column, you eliminate three vertical possibilities, six horizontal possibilities, and six diagonal possibilities. Therefore, the elimination of these 15 total possibilities amounts to 69 combinations of four. It would be very helpful if the constant was named something like VALID\_COMBINATIONS\_OF\_FOUR\_IN\_A\_ROW.