Matthew Martin

998354224

ECS 170, Winter 2017

3/1/2017

Project 2: Connect Four

Team Name: MatthewMartin

1) My evaluation function returns the difference in the number of ways to win for each player. There are 69 ways to win in connect four on a 6x7 board (24 row wins, 21 column wins, 24 diagonal wins), and the evaluation function finds how many ways there are left for each player to win (our ways to win minus opponent's ways to win). The logic to calculate these "ways" is hard-coded in the program, because it gives better performance than running a dynamic calculation. In addition, a winning state returns a large positive number, a losing state returns a large negative number, and a draw-state returns zero. One small detail is that a losing state with more coins (a later state), will return a slightly better evaluation value than an early one. This ensures that, if the AI knows it will lose, it will attempt to lose as late in the game as possible in order to give the opponent more chances to make mistakes. Similarly, it will attempt to win as soon as possible (if winning is inevitable). My evaluation function also values moves that pass through lower middle-positions more highly than other moves (barring the bottom row). This is because winning combinations lower to the bottom of the board are more likely to be executed first (due to the gravity). Row 1 is weighted 4:1, row 2 is weighted 3:1, and row 3 is weighted 2:1.

    a) My evaluation function is based on one of the more obvious evaluation functions; it is intuitive to value the game-state based on the number of winning combinations available to each player. I noticed when playing against Monte Carlo, however, that the pure "winning moves" evaluation functions values moves towards the middle-rows of the board too highly, and will lose to lower-row moves before the middle-row moves are relevant. I believe that adjusting for this via row-weighting will help improve the win-rate of the original intuitive evaluation function. I am hoping that my implementation of the evaluation function with a hard-coded lookup, and implementation of efficient alpha-beta pruning will enable my AI to be competitive among my classmates.

    b) myBlocked: number of my winning ways blocked by opponent's coins (weighted based on the middle-row the "way" passes though)
    opponentBlocked: number of opponent's winning ways blocked by my coins (weighted based on the middle-row the "way" passes through)

    If (game_over_state)
        If (we_win) eval = Integer.MAX_VALUE - number of coins

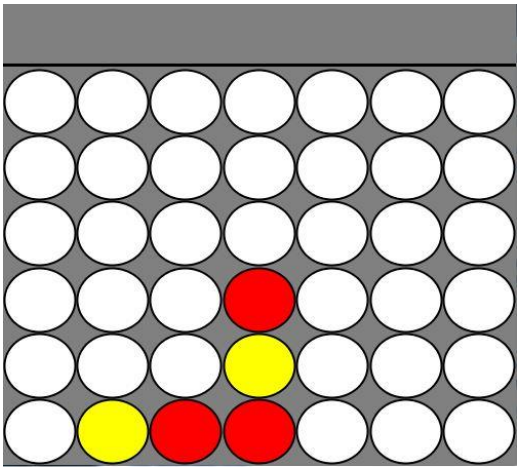
If (opponent_win) eval = Integer.MIN_VALUE + number of coins
If (draw) eval = 0
else
eval = opponentBlocked – myBlocked

c) The board state below has the following evaluation value:
Assuming we are the red player and the opponent is the yellow player:

We are blocking 21 (8 horizontal, 4 vertical, 9 diagonal) of our opponent's possible winning combinations.
Of the 8 horizontal ways, 4 are weighted with a value of 1, and 4 with a value of 3.
Of the 9 diagonal ways, 2 are weighted with a value of 1, 1 is weighted with a value of 4, and 6 are weighted with a value of 3.
Vertical moves are always weighted as 1.
The total value gained from blocking the opponent's moves is thus:
(4*1 + 4*3) + (2*1 + 1*4 + 6*3) + 4 = 44.

The opponent is blocking 14 (6 horizontal, 3 vertical, 5 diagonal) of our possible winning combinations.
Of the 6 horizontal ways, 4 are weighted with a value of 4, and 2 with a value of 1.
Of the 5 diagonal ways, 4 are weighted with a value of 4, and 1 with a value of 3.
Vertical moves are always weighted as 1.
The total value lost from the opponent blocking our moves is thus:
(4*4 + 2*1) + (4*4 + 1*3) + 3 = 40.

The value of the board is thus: 44 – 40 = 4.



2) Submitted minimax_MatthewMartin.java

3) minimax_MatthewMartin.java beats StupidAI.java and RandomAI.java every time.

When my AI goes first, the results are as follows (for 10 arbitrary runs):

Winner	Final Board
Player 1 (minimax_MatthewMartin)	...x... ...xo.. xx.ox.. ooxxx.. oooxo.. ooxoxx.
Player 1 (minimax_MatthewMartin)	...x... .x.xo.. xx.ox.x ooxxx.o oooxoxo ooxoxxo
Player 1 (minimax_MatthewMartin)	....x.. ....x.. ...xx.. o..ox.. o..xo.. ooxox..
Player 1 (minimax_MatthewMartin)	...x... ...xo.. xx.ox.. ooxxx.. oooxo.. ooxoxx.
Player 1 (minimax_MatthewMartin)	...x... ...xo.. xx.ox.. ooxxx.. oooxo.. ooxoxx.
Player 1 (minimax_MatthewMartin)	....x.. ....x.. ...xx.. o..ox.. o..xo.. ooxox..
Player 1 (minimax_MatthewMartin)	...x... .x.xo.. xx.ox.. ooxxx.. oooxo.. ooxoxx.
Player 1 (minimax_MatthewMartin)	...x... ...xo.. xx.ox.. ooxxx.. oooxo.. ooxoxx.
Player 1 (minimax_MatthewMartin)	...x... .x.xo.. xx.ox.x ooxxx.o oooxoxo ooxoxxo
Player 1 (minimax_MatthewMartin)	...x... .x.xo.. xx.ox.. ooxxx..

	ooox.. ooxox..

Won 10/10 games played as player 1.

When Monte Carlo AI goes first, the results are as follows (for 10 arbitrary runs):

Winner	Final Board
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox
Player 1 (MonteCarloAI)	xoooxo. oooxxxx oxoxooo xxxooxx xoxoxxo ooxxoxx
Player 1 (MonteCarloAI)	xoooxo. oooxxxx oxoxooo xxxooxx xoxoxxo ooxxoxx
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox
Player 1 (MonteCarloAI)	xoooxo. oooxxxx oxoxooo xxxooxx xoxoxxo ooxxoxx
Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox

Player 2 (minimax_MatthewMartin)	....oox.. ..oxo.x ..oox.x ..oxxox

Won 7/10 games played as player 2.

Note: minimax_MatthewMartin beats MonteCarloAI a majority of runs as the second player, but not as often as when playing as the first player. As far as I can tell, there is only one winning state and one losing state that appear commonly (these are shown in the table above).

4) My successor function generates children nodes in order of their evaluation functions. If the current node is a max node, it will generate children in order of decreasing evaluation function; if it is a min node it will generate children in order of decreasing evaluation function. Alpha-beta pruning is maximized when the Nash-equilibrium (current "v"-value) for a node's children are in either monotonically decreasing or increasing order (depending on if it's a max or min node). The evaluation function of a node itself should somewhat approximate it's Nash equilibrium (at least in terms of ordering its Nash-equilibrium with respect to its "siblings"). This is because a higher evaluation function for a node will lead to (in general), higher evaluation functions for its children. Thus, ordering children by evaluation function should ensure the best-case scenario.

5) Submitted alphabeta_MatthewMartin.java