Henry Jacobs – hmj32

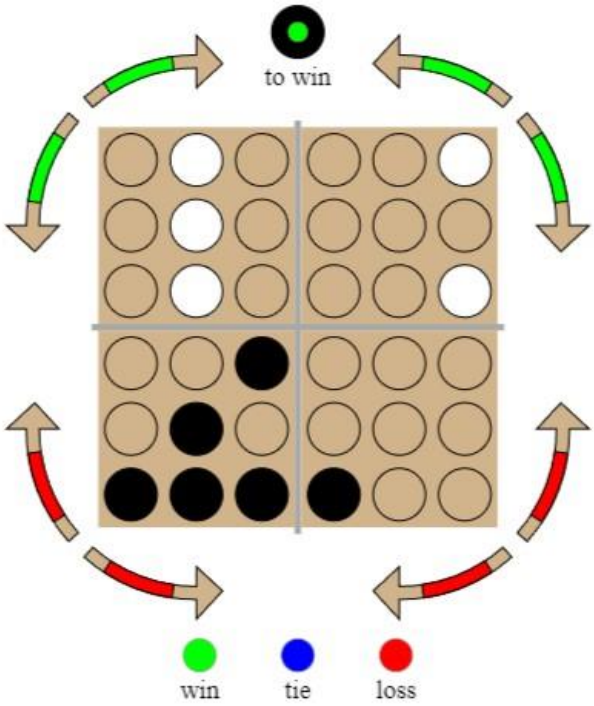
CS 380 – Artificial Intelligence

Professor Popyack

5/31/2022

hmj32_h Pentago Heuristic -- Written Documentation

This heuristic primarily uses three methods in order to compute a score: horizontal_straights, vertical straight, and get_diagonal_score. The heuristic also uses the six for loops nested inside of the final call function to compute a sort-of "multiplier" value for specific states. I will explain all three of these methods below. The image below this paragraph can be used for reference for the explanations.



Black Evaluation Score = 107 (Diagonal += 3, Horizontal += 4, Vertical += 0, 4-in-a-row multiplier += 100)

White Evaluation Score = 3 (Diagonal += 0, Horizontal += 0, Vertical += 3, 4-in-a-row multiplier += 0)

Vertical Straight

This function will calculate all the pieces that are vertically in a row. For example, if we were calculating white's vertical straight score for this board, then they would score a 3 because of the three in-line white pieces in the upper-left quadrant. Vertical straight only awards points for straights with 2 or more pieces in a row. This means that vertical straight will never award points for single pieces lying around, and hence white would not receive any points for the pieces in the upper-right hand quadrant.

The implementation of vertical straight is straightforward. It has two counters that it uses to loop fully through our 2D array (one counter is for the column and the other for the row). It compares every neighboring piece in the first column, then all the neighboring pieces in the second column, etc. until we reach the end of the array. The function returns a value and the number of straights in each column. I will explain more about this in the section about final_call.

Horizontal Straight

Horizontal Straight is like Vertical Straight in that it awards 1 point for each piece that it finds next to another piece of the same type in the same row and does not award points for single pieces lying around (i.e. there must be at least two in a row for horizontal_straight to award points). If we were only calculating the horizontal straight value for black on the board above, we would receive a value of 4 from the four in-line pieces in the sixth row.

The implementation of Horizontal Straight involves two functions: if_token_exists_call_evaluator and evaluator. The first function takes the array returned by calling board.board in the main program, and runs through each of the six-arrays looking for our particular piece. If it happens to find one of the pieces, it then proceeds to pass it into evaluator to be evaluated. Once it's inside of evaluator, it'll just compare the neighboring pieces for the given row. If two pieces are found to be the same and they match the current players token, then it will award a singular point. The if statement that says if (eval >= 1): eval += 1 is used because if four neighbors are found to be the same by the while loop in evaluator, then only three of them will be counted (almost like the first neighbor was ignored), so this is used to make sure that points are accurately awarded per piece. Vertical Straight does something similar with the variable carry_one.

Diagonal Straight

The function Diagonal Straight is also very similar to horizontal straight and vertical straight: it will award one point for each in-line piece that it finds in one of the diagonal rows. Pieces must be in the same row to receive points and there must be more than one piece for points to be awarded. The guts of the diagonal straight function (particularly the while loop) are very similar to that of the horizontal straights function (it compares neighbors and awards one point for each neighbor found. It also uses a carry_one method like Horizontal and Vertical Straight Functions). In order to access the diagonals, I had to hardcode the values into a list. I'm sure there are better ways to do this, but I couldn't think of any within the time frame.

If the Diagonal Straight function were to be applied to the board above, black would receive a value of 3 and white would receive a value of 0.

Final Call

Final Call is responsible for calling all the functions and summing all the scores together to get a composite heuristic score. Values for Diagonals, Verticals, and Horizontal Straights are computed for each player. The current player has their values added to the heuristic and the opponent's values are subtracted from the heuristic. This way, the player can use the heuristic to best understand not only their own position, but that of their opponent. If any of the players are found to have 4 pieces in a row, there is a multiplier applied to their scores. If the current player is found to have a diagonal with 4 in a row and a vertical column with 4 in a row, then the would receive an additional 100 points for each 4-in a row they have (this means they would receive an extra 200 points onto their heuristic). The opposite goes for the opponent, if they had a vertical column and horizontal row with 4 in a row, then 100 points would be deducted from the heuristic. The idea for this is that 4-in a row is a deadly combination and is one move away from a win or a loss, so it's imperative that the player take the right steps to achieve or avoid this scenario based on whichever one is in their favor.

WHAT HAS CHANGED SINCE THE FIRST SUBMISSION

The heuristic remains the same as it was in the first submission, except for one small change which takes place in the six for loops at the end of the heuristic function hmj32_h. The 6 for loops at the end of the function use to only check if the board had contained straight with for in a row and would award the player an extra 100 points for finding such a move. For loops now check if the player and or the opponent has 5 in a row of either a diagonal, horizontal straight or vertical straight. If the current player has five in a row of any of these, the heuristic returns self.INFINITY – 1, while if the opponent has a five in a row straight or diagonal, it awards the player a value of 1 – self.INFINITY.

Without this addition, the heuristic would not have been able to award moves that resulted in a win, seriously hurting its ability to accurately score states.