**Udacity AI Nanodegree: Heuristic Analysis for Advanced Game Playing Agent Module Project**

**Omkar Vedpathak, April 2017 cohort**

The best scoring function I found was the heuristic I’ve implemented as ‘custom\_score’. This heuristic uses one modification on top of the ‘improved score’ heuristic, whose goal is to maximize the difference between the player and the opponent moves. Boards in which the player has more moves relative to the opponent are favored, since this is likely to produce boards where the opponent runs out of moves, which leads to a win for the agent.

The modification I used in addition to the ‘improved score’ heuristic, is to take into account how far away from the center of the board a move is. My custom heuristic will penalize moves that have a larger ‘center\_score’ – i.e. scores which are further from the center of the board. This modification chooses moves that are more central. This is simply to try to favor moves where there are future moves, based on my intuition when attempting to play a sample game on a board. I took the square root of the center score, since it is a squared value by default, and added a scaling factor of 0.5 so its influence on the move utility is similar to the influence of number of moves. E.g. for a piece that is placed in the corner (0,0) at the beginning of the game, the (negative) utility contribution for this move is:

1. 0.5 \* Sqrt((3.5/2-0)2 + (3.5/2-0)2)= 0.5 \* sqrt(3.0625) = 0.5\* 1.75 = 0.875

Which rounds up to 1 – so a move to the corner is equivalent to losing a move. Whereas if a piece is placed more centrally, e.g. at (4,3), the (negative) utility contribution for this move is negligible:

1. 0.5 \* Sqrt((0.5/2-0)2 + (-0.5/2-0)2)= 0.5 \* sqrt(0.125) = 0.5\* 0.353 = 0.17625

The main reason I chose the ‘custom\_score’ heuristic over the other ‘custom\_score’ heuristics I implemented, because it performed experimentally better than the other heuristics overall and better than ‘AB\_Improved’.

‘Custom\_score’ is likely to have outperformed ‘AB\_Improved’ because it favors more central moves which favors the number of branches available and possible moves deeper down in the search tree / after a greater deal of the board is populated.

‘Custom\_score’ performs better than ‘custom\_score\_2’ where I penalized boards in which there were fewer blank spaces. ‘Custom\_score\_2’ likely performed worse than the ‘custom\_score’ because penalizing boards / moves in which there were fewer blank spaces only has the effect of reducing the number of attractive branches in the search tree, which favors neither player. ‘Custom\_score\_2’ performs the same as the AB\_Improved metric, so the effect of including the blank spaces heuristic appears negligible.

‘Custom\_score’ was almost identical to ‘custom\_score\_3’, however the smaller influence of the ‘center\_score’ penalty in ‘custom\_score’ vs ‘custom\_score\_3’ must have contributed to its improved results, since I reduced the effect of the penalty. If the ‘center\_score’ penalty is too large, it is possible that when the board has filled up, the ‘center\_score’ penalty can often lead to fatal moves towards the center where there aren’t many moves left, so the ‘custom\_score’ heuristic models the end game state more effectively than ‘custom\_score\_3’.

All of the heuristics incorporate a mechanism which makes sure that if there is ever a board state where the opponent has no legal moves remaining, and the agent has legal moves remaining, we should guarantee selection of this board state, since it is a winning board state. To guarantee selection of this board state, the utility is set to infinity. The downside of this improvement is that it is rarely actually triggered, since it requires large search depths (>7), since only after a decent number of plies would the opponent ever be in a state where they have no legal moves remaining. The opposite condition is also used to guide the heuristic, i.e. if the player is ever a loser, ensure that moves producing this condition are never selected, so we set the utility to negative infinity.

The time complexity of all of heuristic implementations are linear since we are only counting the number of possible moves of each player, which is always at most 8 per player, since the pieces can only move as Chess knights, and the number of legal moves left per player, to determine if either has won / lost. Getting the number of moves per player twice, still leaves the time complexity as linear, and adding the ‘center\_score’ heuristic penalty is also a linear addition, since we just get the player location which is a constant time lookup.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Match #** | **Opponent** | **AB\_Improved** | | | **AB\_Custom** | | | **AB\_Custom\_2** | | | **AB\_Custom\_3** | | |
|  |  | **Won** | | **Lost** | | **Won** | **Lost** | | **Won** | **Lost** | | **Won** | **Lost** |
| **1** | **Random** | 9 | | 1 | | 9 | 1 | | 10 | 0 | | 10 | 0 |
| **2** | **MM\_Open** | 7 | | 3 | | 8 | 2 | | 5 | 5 | | 5 | 5 |
| **3** | **MM\_Center** | 9 | | 1 | | 8 | 2 | | 9 | 1 | | 9 | 1 |
| **4** | **MM\_Improved** | 7 | | 3 | | 8 | 2 | | 7 | 3 | | 8 | 2 |
| **5** | **AB\_Open** | 4 | | 6 | | 7 | 3 | | 4 | 6 | | 5 | 5 |
| **6** | **AB\_Center** | 7 | | 3 | | 5 | 5 | | 7 | 3 | | 4 | 6 |
| **7** | **AB\_Improved** | 5 | | 5 | | 5 | 5 | | 6 | 4 | | 4 | 6 |
|  | **Win Rate %** | 68.6 |  | | | 71.4 |  | | 68.6 |  | | 64.3 |  |